DEFENSE INFORMATION INFRASTRUCTURE (DII)

COMMON OPERATING ENVIRONMENT (COE)

PROGRAMMER'S MANUAL

Version 2.0

28 June 1996

Prepared for:

Defense Information Systems Agency

Table of Contents

Table of Contents (continued)

Table of Contents (continued)

List of Figures

This page intentionally left blank.

SECTION 1.0 INTRODUCTION

1.1 Purpose

The *DII COE Programmer's Manual* is a "living" source document for information about the Defense Information Infrastructure (DII) Common Operating Environment (COE). It was created, in conjunction with the *DII COE Programmer's Reference Manual*, to support programmers' development activity in the DII common operating environment. This manual provides an overview of the COE design and development concepts, discusses the requirements and standards that apply in this environment, describes a methodology for open system development, and addresses software component structure and selected toolsets. Additional information regarding this specific DII COE release is also included. The concept for the *DII COE Programmer's Manual* is to provide new COE users and programmers with a manual for system and software development within the DII COE environment.

The Defense Information Systems Agency (DISA) will review and update this document as required to remain current with the evolution of the DII COE. This document supersedes Version 1.0, all earlier draft versions, presentations, or working group notes. Please direct any comments or questions regarding the *DII COE Programmer's Manual* to:

Point of Contact (POC)	Cmdr. Charles B. Cameron Chief, DII COE Engineering Division
Address	DISA/JIEO/JEXF-OSF 45335 Vintage Park Plaza Sterling, VA 20166-6701
Telephone	(703) 735-8825
Fax	(703) 735-8761
Internet Address	cameronc@ncr.disa.mil

1.2 Background

The DII is a Department of Defense (DoD) enterprise-level effort to develop and field military systems that will meet the needs of the warfighter in a global information environment. The DII provides a common operating environment for all DoD information systems domains. The objective of the DII COE is to furnish a common application environment that satisfies the needs of DoD application developers and supports individual mission area requirements.

1.3 Scope

This document describes the concepts, models and architecture as well as the technical requirements for building and integrating software components on top of the DII COE. It is comprised of ***segments*** which are collections of one or more Computer Software Configuration

Items (CSCI). This document provides implementation concepts with details which describe, from a software development perspective, the following:

- the Common Operating Environment approach to software reuse,
- the runtime execution environment for programmers,
- the requirements for COE applications compliance,
- how to structure components to automate software integration, and
- how to electronically submit/retrieve software components to/from the software repository.

All segments submitted to DISA must be in compliance with this document.

1.4 Document Structure

This *DII COE Programmer's Manual* is divided into seven sections. Each section is devoted to a topic of primary importance in DII COE development activity.

Section 1 discusses the purpose, background, and scope of this document. It also contains an overview of the DII COE.

Section 2 describes the requirements for the DII COE to support DoD services and mission applications, discusses COE design and development considerations, and introduces the *Integration and Runtime Specification* (I&RTS) document.

Section 3 describes the COE Kernel, introduces the COE runtime environment, addresses development concerns that must be considered to reduce the risk when integrating or configuring systems and/or software, introduces the tools available to assist the COE application developer, and introduces Applications Program Interfaces (API) and discusses the technical requirements and Application Portability Profiles (APP).

Section 4 identifies elements of delivered COE components.

Section 5 discusses segment compliance within the COE, outlines the process of registering segments in the COE, discusses configuration management for the DII, and outlines the procedures for reporting DII COE problems.

Appendices There are six appendices (A through F) which contain the following:

A. Acronym List
B. Glossary
C. References
D. DII COE Hardware/Software Platforms
E. On-Line Information

SECTION 2.0 COE ARCHITECTURE OVERVIEW

2.1 Overview

The DII COE concept is best described as an architecture that is fully compliant with the *Department of Defense Technical Architecture Framework for Information Management* (TAFIM). The COE consists of architecture, standards, reusable software components, and a software repository library.

In COE-based systems all software, except the operating system and basic windowing software, is packaged in self-contained units called *segments*. Segments are the most basic building blocks and are defined in terms of the functions they perform. They may contain one or more Computer Software Configuration Items (CSCI). Reusable segments that are part of the COE are known as COE components or simply "components." The principles governing how segments are loaded, removed, or interact with one another are the same for all segments. COE component segments are treated more strictly because they form the foundation for the COE systems. Selecting software modules that fulfill the COE component requirements, and extending them to meet other problem domains is an ongoing task. The COE preserves backward compatibility so that mission applications are not abandoned just because there is an update of the COE.

2.1.1 The DII COE Concept

The DII COE concept provides a distributed application infrastructure that promotes interoperability, portability, and scalability DoD-wide. Its objective is to provide access to data, independent of location, in a reliable, cost-efficient manner. The DII COE forms an open system foundation that is presently used in the Global Command and Control System (GCCS) and the Global Combat Support System (GCSS). Each system built upon the COE foundation uses the same set of APIs to access common COE components, the same approach to integration, and the same set of tools for enforcing COE principles. Precisely the same COE software components are used for common functions, such as communications interfaces and data flow management.

Although the DII COE model emphasizes both software reuse and interoperability, its principles are more far reaching and innovative. The COE concept encompasses:

- an architecture and approach for building interoperable systems,
- an infrastructure for supporting mission area applications,
- a rigorous definition of the runtime execution environment,
- a rigorous set of requirements for achieving COE compliance,
- an automated toolset for enforcing COE principles and measuring COE compliance,
- an automated process for software integration,
- a collection of reusable software components,
- an approach and methodology for software reuse,
- a set of APIs for processing COE components, and
- an electronic process for submitting/retrieving software components to/from the COE software repository.

This document first and foremost describes ***how*** modules must interact in the target system. System architects and software developers retain considerable freedom in building the system, but runtime environmental conflicts are identified and resolved through automated tools that enforce COE principles. An important side effect is that traditional integration tasks become the responsibility of the developer. Developers must integrate and test their software within the COE prior to delivering it to the government. This simplifies integration because those who best understand the software design (the original developers) perform it. It reduces the cost because integration is performed earlier and at a lower level in the process, and it allows the government to concentrate on validation instead of integration.

In the context of this document, the COE must be understood as a multifaceted concept. Proper understanding of how the many facets interact is important in appreciating the scope and power of the DII COE and in understanding the COE concept. The next subsection deals with three specific facets in more detail: the COE as a system foundation, the COE as an architecture, and the COE as an implementation strategy.

2.1.2 The DII COE Foundation – OSE

The COE is an Open System Environment (OSE) that provides the *foundation* for building an open system and provides a practical update mechanism for operational sites. GCCS is the first system built using the DII COE while GCSS is in progress.

2.1.2.1 DII COE as a System Foundation

Figure 2-1 shows how the DII COE serves as a foundation for building multiple systems. The diagram shows two types of reusable software: the operating system, and the COE components. Section 2 of the *I&RTS* describes the COE components in more detail, and the supported operating systems. It is sufficient to note that these components are accessed through APIs and that they form the architectural backbone of the target system.

Building a target system, such as GCCS or GCSS, is largely a matter of fielding new COE components and combining new and existing COE components with mission-specific software. The COE infrastructure manages the flow of data through the system, both internally and externally. Mission-specific software is primarily concerned with requesting data from the COE and then presenting it in a form that is most meaningful to the operator (e.g., as a pie chart, in tabular form, as a graph). The COE provides the necessary service primitives for such data manipulation, and contains information about where the requested data is stored, whether locally or remotely across the LAN/WAN. This frees the system designer to concentrate on meaningful data presentation rather than on the mechanics of data manipulation, network communications, database storage, etc.

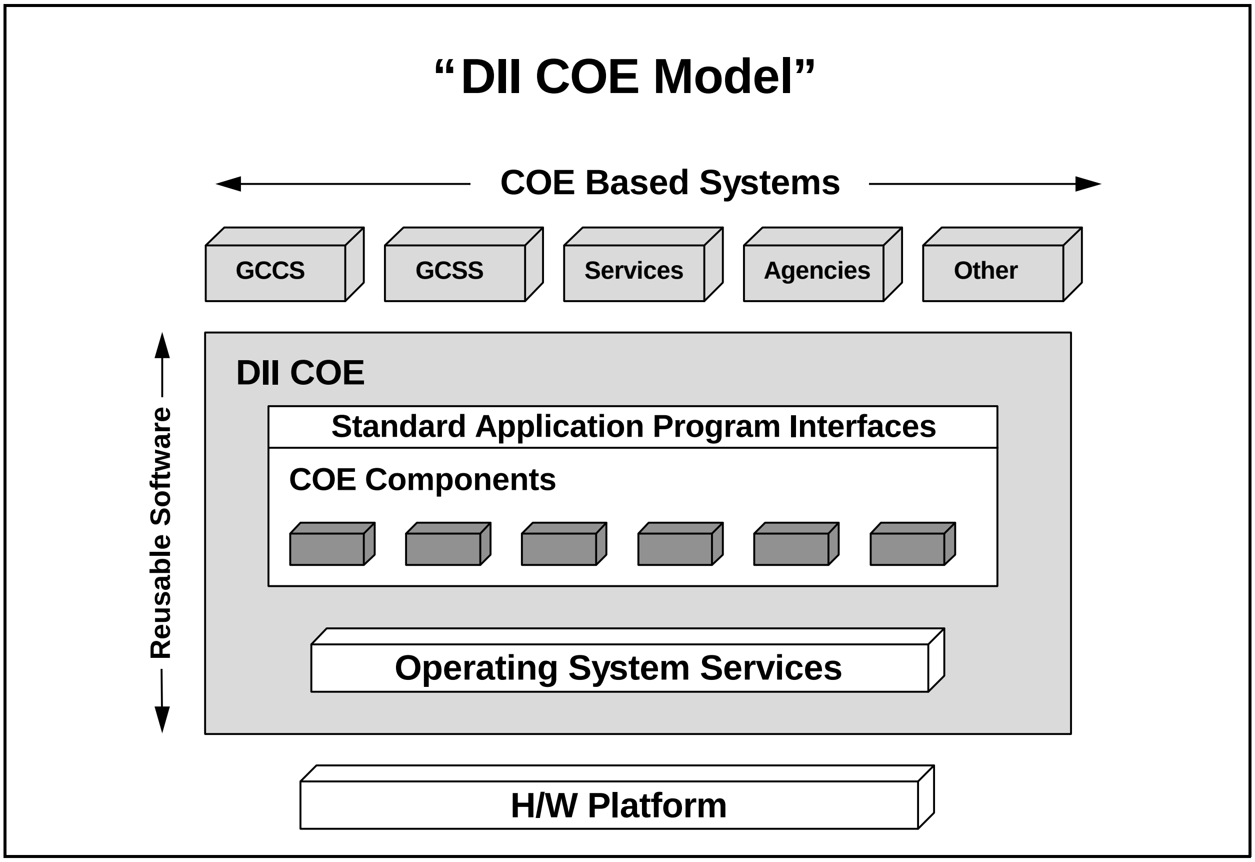


Figure 2-1. DII COE and COE Based Systems

It is important to note that there is only one COE. Each system uses the same set of APIs to access common COE components, the same approach to integration, and the same set of tools for enforcing COE principles. Systems are built on top of the COE and use precisely the same COE components, not just the same algorithms, for common functions (e.g., communications interface, data flow management, etc.). This approach to software reuse significantly reduces interoperability problems because, since the same software is used, it is not possible to interpret or implement standards differently.

2.1.2.2 DII COE as an Architecture

The DII COE is an open architecture that is not tied to a specific hardware platform. It uses POSIX-compliant operating systems and industry standards such as X-Window and Motif. The present COE taxonomy and architectural design requirements are detailed in the *Architectural Design Document for the Defense Information Infrastructure (DII) Common Operating Environment (COE)*, January 1996.

The DII COE architecture is divided into two major areas: Platform Services and Common Support Applications. Platform Services are those types of services that support the flow of information, while Common Support Applications do not directly support the flow of information but are critical to interoperability.

Platform Services include the following functions:

- Management Services - enables applications and system functions to be monitored. These include network and system administration services.

- Security Services - provides support for three major information security requirements: confidentiality, integrity, and availability.

- Communications Services - provides various modes of communication between parts of distributed applications.

- Distributed Computing - allows geographically distributed applications and services to interoperate as if they were all located on the same computing platform.

- Data Management Services - provides definition, storage, and retrieval of files, databases, and object bases distributed over the network. Includes basic management functions to maintain operational integrity and application availability.

- Presentation Services - provides end-user and output support services needed for interoperability across local and wide area networks. These include display (window) management, shared input and output device management (printers, plotters, and scanners), and support for the interchange of multimedia (voice, video, imagery, and animation).

Common Support Applications include the following functions:

- Alert Services - responsible for the routing of alert and system event messages. Also provides ability to dynamically define alerts and other system events, and the system action related to these events.

- Correlation Service - responsible for maintaining a consistent collateral-level tactical picture across a theater of operations based on information provided by all service components. Provides inputs to the tactical plotting functions of the Mapping, Charting, Geodesy, and Imagery (MCG&I) Service.

- MCG&I Service - provides common geospatial processing and data to all mission applications and users.

- Message Processing Service - responsible for the management and timely distribution of messages in text and binary formats between files, processes, databases, users and devices.

- Office Automation - provides general purpose capabilities for end-users, including word processing, electronic mail, presentation graphics, electronic spreadsheet, and drawing packages.

2.1.2.3 COE as an Implementation Strategy

The COE is also an evolutionary acquisition and implementation strategy. This represents a departure from traditional development approaches. It emphasizes incremental development and fielding to reduce the time required to put new functionality into the hands of the warrior, while maintaining software quality and minimizing program risk and cost. This approach is sometimes described as a "build a little - test a little - field a lot" philosophy. It is a process of continually evolving a stable baseline to take advantage of new technologies as they mature and to introduce new capabilities. The changes are made one step at a time so that the users always have a stable baseline product while changes between successive releases are perceived as slight. Evolutionary development has become a practical necessity for many development programs because the traditional development cycle time is longer than the technical obsolescence cycle time.

From the perspective of a COE-based system, the implementation strategy is to field new releases at frequent intervals. Each release might include enhancements to both the COE and mission-area applications. Mission-area applications are considered to be provisional, subject to user feedback. Applications for which feedback is favorable are retained in subsequent releases and hardened as needed for continued operational use. As appropriate, mission applications that are widespread in use and commonality will be integrated into the COE or enhanced to add new features.

The COE implementation strategy is carefully structured to protect functionality contained in legacy systems so that over time they can migrate to full COE utilization. This is achieved through publishing "public" and "private" APIs. Public APIs are those interfaces to the COE that will be supported for the life-cycle of the COE. Private APIs are those interfaces that are supported for a short period of time to allow legacy systems to migrate from unsanctioned to sanctioned APIs. All new development is required to use only public APIs; use of any other APIs results in a non-COE-compliant segment. The process of migrating from existing legacy "stove-pipe" systems to utilizing the COE is a primary reason for articulating technical requirements for the COE, and it provides program managers with information useful for establishing development priorities.

DII uses COE system administration tools to allow site administrators to selectively install only those software applications required for the site. This minimizes hardware requirements and simplifies site administration. Site administrators can further tailor the installation so that operators are given access to only those applications that pertain to their area of responsibility.

Software updates will be available periodically as new capabilities are developed, or as software patches are created to fix problems. Site administrators can receive these updates via tapes, or electronically across the Secure Internet Protocol Router Network (SIPRNet). Electronic updates are available, in either a "push" (e.g., the update process is initiated electronically by a DISA Software Support Activity) or "pull" mode (e.g., the update process is initiated electronically by the operational site).

2.2 Model & Principles

This section will introduce the DII COE Concepts, Model and Structure of the development environment. It also provides a guidance on conformance assessment for the developer's applications and the products that comprise the DII COE. Additional detail information on the tools and APIs for COE are documented in the *DII COE Programmer's Reference Manual*.

Figure 2-2 is a simplified diagram that illustrates the relationship between the COE, component segments, and mission application segments. The COE encompasses APIs, Government-Off-The-Shelf (GOTS) and Commercial-Of-The-Shelf (COTS) software, the operating system, windowing software, and standards (see the *User Interface Specifications for the Defense Information Infrastructure (DII)*, also referred to as the *DII COE Style Guide*, the *DII COE Integration and Runtime Specification* (I&RTS), etc.). Physical databases (e.g., MIDS, IDB, JOPES database, etc.) are not considered part of the COE although the software, such as the relational database management system (RDBMS) which accesses and manages the data, is part of COE. Figure 2-2 is a generic diagram intended only to show relationships and relative position of the services, APIs and the mission applications. The labeled boxes show services useful for the C4I, logistics and mission applications domains while the principles remain unchanged. The diagram remains applicable even when service boxes are replaced with examples of another problem domain.

To use a hardware analogy, the COE is a collection of building blocks which form a software "backplane." Segments "plug" into the COE just as circuit cards plug into a hardware backplane. The blocks containing the operating system and windowing environment are akin to a power supply because they "power" the rest of the system. The segments labeled as COE component segments are analogous to already-built boards such as CPU or memory cards. Some of them are required (e.g., CPU) while others are optional (e.g., specialized communications interface card) depending upon how the system being built will be used. The blocks along the top of Figure 2-2 labeled as "mission application areas" are composed of one or more mission application segments. These segments are analogous to adding custom circuit cards to the backplane to make the system suitable for various purposes.

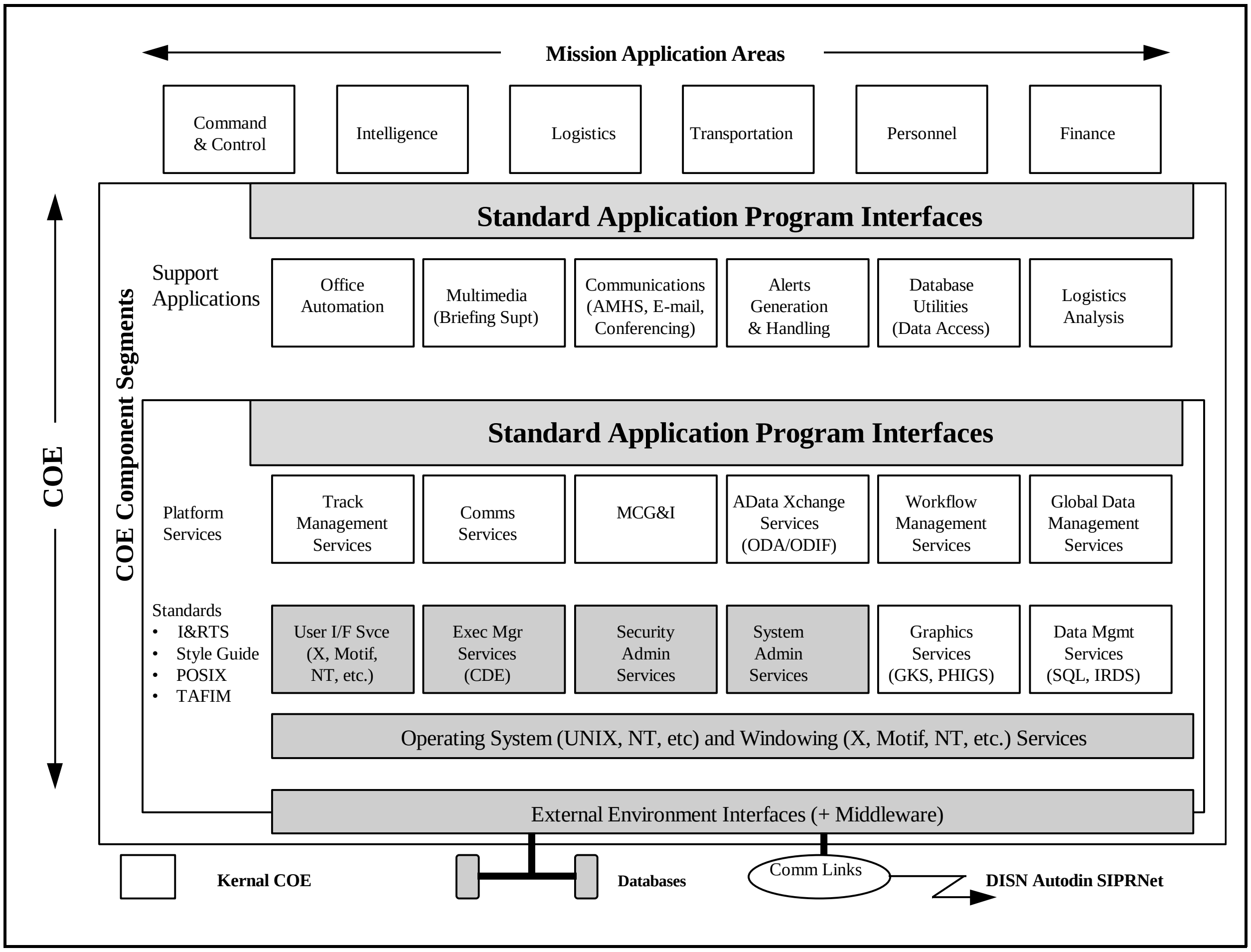


Figure 2-2. DII COE Model

2.2.1 DII COE Model

The DII COE APIs define how other segments may connect to the backplane and utilize the electric "power supply" or other circuit cards. This is analogous to a hardware schematic diagram that indicates how to build a circuit card that will properly plug into the backplane. Figure 2-2 also implies that APIs are the only avenue for accessing support application and platform services provided by the COE. This is true for all COE software, including COTS software. However, the COE does not create an additional layer on top of the COTS software. These components may be accessed directly using vendor-supplied APIs for these commercial products as long as such usage does not circumvent the COE model. For example, the COE includes a POSIX-compliant operating system. Some vendors provide non-POSIX compliant extensions to the operating system services. Use of such extensions, even when readily available through vendor supplied APIs, is not allowed because such usage violates the COE architecture and inhibits interoperation, portability and reuse.

This hardware analogy can be extended to implementation of databases within the COE, but with some significant distinctions. Within this conceptual model, the DBMS functions as the COE's disk controller and disk drives. The applications databases, which are not part of the COE, can be equated to directories or partitions on the drives accessed through the DBMS disk controller. Data objects belonging to each database can be considered as files within those directories.

This analogy is critical to understanding the modularity limitations for a database within the COE. One can change out most peripherals or circuit cards without any side effects just as one can replace mission applications without losing information. However, one cannot install a larger disk drive or change from one type of controller to another without losing the data on the disk. Although upgrading mission applications is like swapping circuit cards, upgrading databases is like rebuilding a disk or directory structure. Instead of replacing a component, one must save and then restore the files on the disk. Further, each site's database must be presumed to be site-specific in the same way that disks in different systems have different files.

COE databases are divided among segments as are mission applications, but with a different focus. Mission applications are segmented based on their functionality. Databases are segmented by their content or by the subject area of the mission applications they support. Thus mission applications are functional modules; databases are informational modules.

The precise configuration of COTS products used in the COE is under strict configuration control. This is necessary because configurable items such as the amount of shared memory or swap space must be known and carefully controlled in order for other components in the COE to operate properly. For this reason, COTS products are assigned a DII COE version number in addition to the vendor-supplied version number so as to be able to track and manage configuration changes.

A fundamental principle throughout the COE is that segments are not allowed to directly modify any resource "owned" by another segment. This includes files, directories, modifications to the operating system, and modification to windowing environment resources. Instead, the COE provides tools through which a segment can request extensions to its base environment. The importance of this principle cannot be overemphasized because environmental interactions between software components are a primary reason for integration difficulties. By providing software tools which arbitrate requests to extend the environment, integration can be largely automated or at least potential problem areas can be automatically identified.

For example, the COE predefines a set of ports in the UNIX `/etc /services` file. Some segments may need to add their own port definitions, but this will create conflicts if the port definitions are the same as those defined by the COE or another segment. To identify and prevent such conflicts, segments issue a request to the COE to add their port definitions. This process is called "environment extension" because a segment is modifying the predefined environment by extension, not through replacement or deletion.

COE component segments shown in Figure 2-2 (DII COE Model) are typically designed to be servers. However, note that in practice such segments will often operate in both a client and server mode. For example, a track management segment is a server for clients that need to retrieve the current latitude/longitude location of a platform. But the track manager itself is a client to a communications server which initially receives track related reports from sensors or other sources. Refer to the *Architectural Design Document for DII Common Operating Environment (COE)* for more detailed discussion of how the COE component segments are designed and interact. For purposes of the present discussion, it is sufficient to view COE segments as servers which are accessible through APIs.

The COE developer's environment provides a variety of segments, not all of which are required for every application. The *kernel COE* is the mandatory minimal set of software required on every workstation regardless of which workstation will be used. The kernel COE components are the shaded boxes in Figure 2-2 (DII COE Model). A kernel COE will always contain the operating system, windowing services, and external environment interfaces. It will also include features as follows:

2. a basic System Administration function,
2. a basic Security Administration function,
2. an Executive Manager function (e.g., a desktop GUI),
2. a template for creating privileged operator logging accounts, and
2. a template for creating non-privileged operator logging accounts.

The System Administration segment is required because it contains the software necessary to load all other segments. The Security Administration segment is required because it enforces the system security policy. The Executive Manager is required because it is the interface through which an operator issues commands to the system. The Executive Manager is an icon and menu driven desktop interface, not a command line interface. The templates included in the kernel COE describe the basic runtime environment context that an operator inherits when he/she logs into the system. This login defines which processes run in the background, which environment variables are defined, etc. The kernel COE assures that every workstation in the system operates and behaves in a consistent manner, and that every workstation begins with the same basic components within the systems environment.

2.2.2 COE Principles

Selection of the specific software modules which comprise the COE determines which problem domain(s) can be addressed by the particular COE installation. However, selection of COE components is not arbitrary. It is driven by a number of important architectural **and** programmatic principles. First, there is a determination of what functions the COE is required to perform. There is a set of criteria for selecting software components which perform the required functions. A function is part of the COE if it meets one or more of the following criteria:

2. **The function is part of the minimum software required to establish an operating environment context.** This is normally provided by COTS products and includes UNIX, X Windows, Motif, security software, and networking software.

2. **The function is required to establish basic data flow through the system.** To be useful, a system must have a means for communicating with the external world. To be efficient, consistent, and robust, a system must also have standard techniques for managing data flow internal to the system. The COE contains these types of

functions. For example, a COE appropriate for a mission application must include communications interfaces, message processing, track management, correlation, tactical display management, database management software, alerts management, and cartographic display functions.

2. **The function is required to ensure interoperability.** Standards alone cannot guarantee interoperability. However, using common software for common functions leads to achieving end-to-end applications interoperability.

2. **The function is of such general utility that if rewritten it constitutes appreciable duplicative effort.** This includes printer services, a service for disseminating alerts, and a desktop environment for launching operator initiated processes.

The first three criteria listed above are technical in nature because they dictate from an architectural perspective what software **must** be contained in the COE for a given problem domain. The fourth criterion, however, is more programmatic because it is often a tradeoff between the cost of modifying a legacy system to remove duplication versus the cost of:

- maintaining duplicate code,
- potentially requiring additional hardware resources because of duplication, and
- operator training when there are different ways to accomplish the same action.

COE compliance requires that there be no duplication of functions in the first three criteria, but some flexibility is possible for the fourth.

The COE is designed to constrain the modification or customization of included modules. The COE components must be carefully configured and managed to provide a controlled way to formally respond to reported problems. Stability of the COE is crucial to the system, so modifications are done carefully, deliberately, and at a slower pace than changes in non-COE routines. However, just because changes are controlled does not mean that the COE routines can not be customized. Ongoing work in the COE is to devise and refine techniques to “open up the architecture.” This will enable applications to customize COE components in ways that do not violate COE principles or adversely impact other developers using COE components and services.

The COE is not designed as a single monolithic process, but is instead designed as a collection of relatively small processes to maintain flexibility and maintain performance. A small number of these are loaded into memory as background processes. Most are loaded into memory on demand in response to operator actions (e.g., edit a file, display a parts inventory, etc.) and only for the amount of time required for them to perform their task. This approach offers considerable flexibility because it limits the number of background processes required. With the exception of adding new background processes, performance is not adversely impacted by adding new segments. The price paid is a small amount of overhead required to load functions on demand, but this is generally negligible because the overhead is small and is usually in response to an operator request to bring up a display that must respond only at human speeds.

A COE component can be omitted from the installed COE if the functionality provided by the segment is:

- not required by any mission application,
- not required by any remaining COE component segments, or
- duplicated by other segments.

A common pitfall to avoid is omitting a COE component because its functionality is available through some other means. The problem with this approach is that a common "look and feel" and consistent operation is no longer presented between applications, and interoperability may be at risk. Omission of COE component segments which are not used is normally handled automatically by the COE installation software.

2.3 Architecture

The DII COE is structured as a "plug and play" architecture. The key to the 'plug and play' design is conformance to the COE through the rules detailed in this document, and through using only the published APIs for accessing COE services. There is considerable danger in using unpublished, proprietary (private) APIs, or APIs from legacy systems, because there is no guarantee that they will remain the same or even exist in subsequent releases. This is no different than using vendor supplied COTS products. The risks are the same.

Discussion of the COE as a "plug and play" architecture is not intended to trivialize the effort which may be required to develop and integrate a segment into the COE. Migration of existing legacy systems to the COE is conceptually straightforward, but may require considerable effort due to the requirement to switch to a different set of building blocks. That is, the effort may not be so much in conforming to a new architectural concept but in modifying code to use a different set of APIs. The 'plug and play' concept clearly conveys the goal and the simplicity that the majority of segment developers will encounter.

2.3.1 Building Blocks - Segments

In COE-based systems, all software except the operating system and basic windowing software is packaged in self-contained units called *segments.* This is true for COE infrastructure software and for mission application software as well. Segments are the most basic building blocks from which a COE-based system can be built. Segments are defined in terms of the functionality they provide, not in terms of "modules," and consist of one or more CSCIs. Defining segments in this way is more natural in expressing and communicating what software features are to be included in or excluded from the system, rather than defining them by individual processes or file names.

Those segments which are part of the COE are known as COE component segments, or more concisely, as segments which further have the attribute of being contained within the COE. The principles which govern how segments are loaded, removed, or interact with one another are the same for all segments, but COE component segments are treated more strictly because they are the foundation on which the entire system is based.

It is important to note and understand that just because a segment is part of the COE, it is not necessarily always present or required. Considerable flexibility is offered to customize the environment so that only the segments required to meet a specific mission application need are present at runtime. This approach allows minimization of hardware resources required to support COE-based systems. For example, the COE contains a service for displaying maps. However, some C4I operators in command centers only need to read and review message traffic and do not need or want to view a tactical display. Logistics operators, using DII COE for the GCSS, do not need to see the tactical picture at all and may only desire to see a map when planning transportation routes. For such operators, it is not necessary at runtime to have the extra memory and performance overhead of the segments which generate cartographic displays.

2.3.2 Interoperability and Integration

It is also important to distinguish between interoperability and integration. In the context of this document, *interoperability* refers to the ability for two systems to exchange data:

- with no loss of precision or other attributes,
- in an unambiguous manner,
- in a format understood by both systems, and
- in such a way that interpretation of the data is precisely the same.

Two systems which are interoperable can exchange data and will produce exactly the same "answer" in the presence of identical data. It is sometimes useful to describe segments as interoperable, but generally the concern is with system-level interoperability. Within the COE system interoperability is achieved only when systems use exactly the same software module to perform identical functions. Implementation of agreed upon paper standards is not sufficient by itself.

Integration, within the context of this document, refers to combining segments, not systems. Segment integration refers to the process of ensuring that segments:

- work correctly with the COE runtime environment,
- do not adversely impact one another,
- conform to the standards described in this document,
- have been validated by the COE tools, and
- can be installed on top of the COE by the COE installation tools.

Integration does not imply interoperability - it only provides a level of assurance that the system will work as designed. Integration of a segment with the COE is the responsibility of the segment developer. Integration of the system as a whole and interoperability testing are performed by government integrators.

2.4 Taxonomy and Foundations

The present COE taxonomy is detailed in the *Architectural Design Document for the Defense Information Infrastructure (DII) - Common Operating Environment (COE).* COE approaches the taxonomy from a perspective of providing a collection of Infrastructure Services and a collection of Common Support Applications. These are summarized in Figure 2-3 and extended to include logistics and mission support services. This taxonomy approaches the problem from an architectural perspective rather than functional.

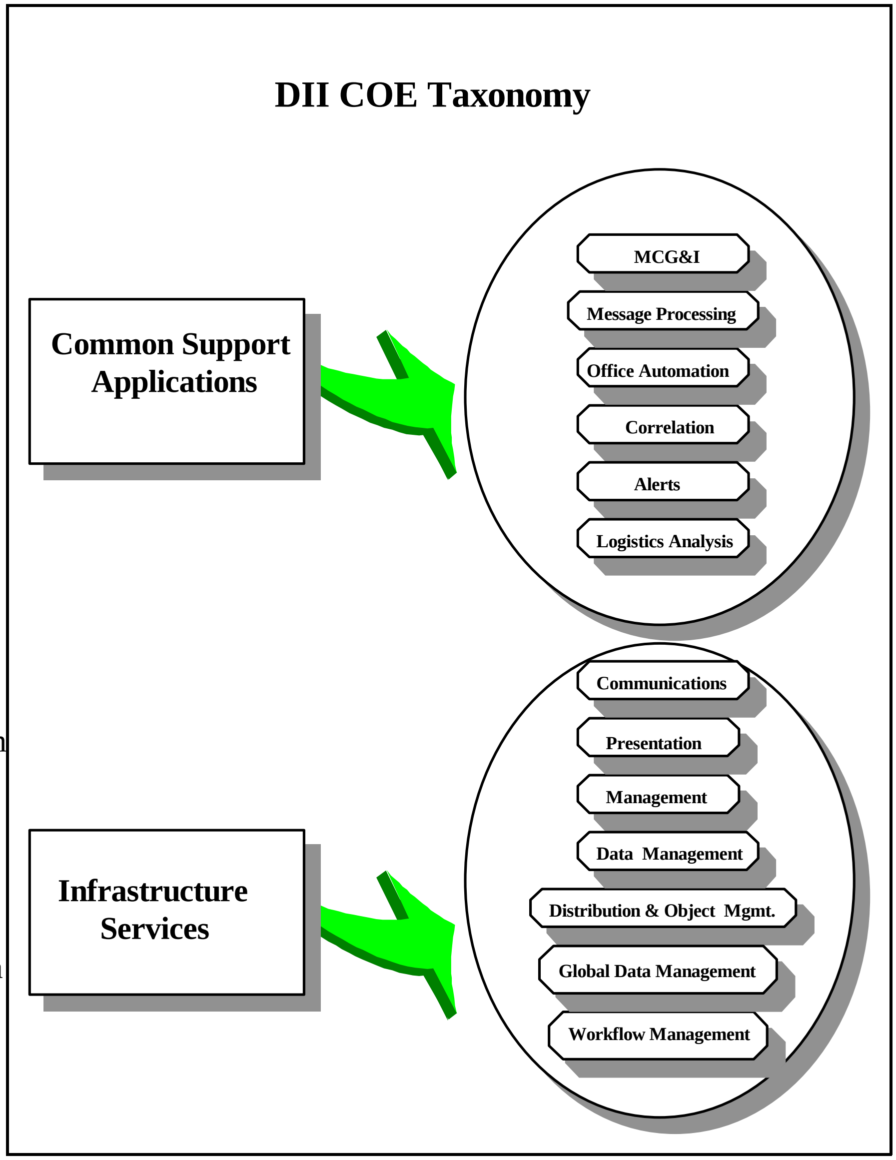


Figure 2-3. DII COE Taxonomy

Infrastructure Services provide the architectural framework for managing and distributing the flow of data throughout the system. Management Services include network, system, and security administration. Communications Services provide facilities for receiving data external to the system, and for sending data out of the system. Distribution and Object Management Services provide the infrastructure necessary to achieve true distributed processing in a client/server environment. Data Management Services include relational database management as well as file management in a distributed environment. Presentation Services are responsible for direct interaction with the human by means of using windows, icons, menus, command or multi-media. Workflow and Global Data Management Services are oriented toward managing logistics data (parts inventory, work in process, operations processes, etc.).

Although Infrastructure Services originated from a problem domain, they are independent of any particular application. Common Support Applications tend to be much more specific to a particular problem domain. The Alerts Service is responsible for routing and managing alert messages throughout the system whether the alert is an “out of paper” message to a systems administrator or an “incoming missile” alert to a watch operator. The Correlation Service is responsible for maintaining a consistent view of the battle space by correlating information from sensors or other sources that indicate the disposition of platforms of interest. MCG&I Services handle display of Defense Mapping Agency maps or other products, and imagery received from various sources. Message Processing Services handle parsing and distribution of military format messages (e.g., Defense Message System (DMS)). Office Automation Services handle office suites such as word processing, spreadsheet, briefing support, electronic mail, World-Wide Web browsers, and other related functions. Logistics Analysis contains common functions, such as Pert charts, for analyzing and displaying logistics-related information.

Selection of software modules which fulfill these COE component responsibilities, and extending them to meet the requirements of other problem domains, is an on-going effort consistent with the evolutionary nature of the COE. Even though the process is evolutionary, the COE preserves backwards compatibility so that mission applications are not abandoned just because there is an update of the COE. Refer to the appropriate API, User’s Guide, and system release documentation for detailed information on the components currently selected in the COE.

2.5 Design and Development Considerations

DII COE is a system specifically designed to meet the C4I information processing needs at various levels of DoD. It consists of geographically distributed workstations interconnected via LAN/WAN technologies. The features provided and the LAN/WAN topology allow users to collaboratively share mission responsibilities. Collaboration is possible in areas as diverse as creating Time Phased Force and Deployment Data (TPFDD), distributing Air Tasking Orders (ATO), performing intelligence analysis, and maintaining a common view of the battlefield with up-to-date display of the deployment of all joint and enemy forces.

This section covers the essential elements for the design and development of mission applications using the DII COE tools and APIs. This is not intended to be a comprehensive guide for programming. Rather, it provides the concepts and considerations in the design of client/server-based distributed applications for DII. For more comprehensive design and development for life-cycle maintenance, refer to the Software Engineering Institute (SEI) Capability Maturity Model (CMM).

2.5.1 System Concepts for Programming

The DII *system* effectively integrates people and organizations, data and information, and work processes and computing services across the DoD.

The DII COE is a “plug and play” open architecture designed around a client/server model. Segments provide functionality that is easily added to or removed from the target system in small manageable units. Segments are defined in terms of functions that are meaningful to operators, not in terms of internal software structure. Structuring the software into segments in this manner is a powerful concept that allows considerable flexibility in configuring the system to meet specific mission needs or to minimize hardware requirements for an operational site. Site personnel perform field updates by replacing affected segments through use of a simple, consistent, graphical user interface.

The DII COE model is analogous to the Microsoft Windows paradigm. The idea is to provide a standard environment, a set of standard off-the-shelf components, and a set of programming standards that describe how to add new functionality to the environment. The Windows paradigm is one of a "federation of systems" where properly designed applications can coexist and operate in the same environment. However, coexistence is not enough. It must be possible for applications to share data. The DII COE extends the Windows paradigm to allow for true integration of systems because mission applications share data at the server level.

Integration has an important architectural advantage over federation. However, integration is not possible without strict standards that describe how to properly build components to add to the system. This document, and related documents, detail the technical requirements for a well behaved, COE-compliant application. The DII COE provides automated tools to measure compliance and to pinpoint problem areas. A useful side effect of the tools and procedures is that software integration is largely an automated process, thus significantly reducing development time while automatically detecting potential integration and runtime problem areas.

More precisely, to a developer the DII COE is:

An Architecture: A TAFIM-compliant client/server architecture which precisely defines how system components will interact and fit together, thus providing a definition of the system level interface to the COE components.

A Runtime Environment: A standard runtime operating environment that includes "look and feel," operating system and windowing environment standards. Since ***no*** single runtime environment is possible in practice, the COE architecture provides facilities for a developer to extend the environment but not to conflict with other developers or system components.

Software: A clearly defined set of existing software within a repository providing already implemented, reusable functions.

APIs: A collection of APIs for accessing COE components. Thus the COE is a set of building blocks in the same sense that X Window and OSF/Motif are building blocks for creating Graphical User Interface (GUI) applications.

DISA maintains the software in an on-line configuration management repository called COE Software Repository System (CSRS). This decreases the development cycle by allowing developers to receive software updates and to submit new software segments electronically.

2.5.2 Building for Integration

Integrated applications are the main focus of the DII mission because of their importance to the DoD efforts in applications interoperability, portability, scaleability, and component reuse. Integration of mission applications is paramount to system operations and the user community within the DII. However, systems applications components within a platform require a high level of integration to assure component independence of software versions, enhancements and upgrades.

2.5.2.1 Rightsizing of Applications

"Rightsizing" requires the optimum division and allocation of computing tasks between client and server for distributed applications. Rightsizing also takes into consideration the requirements for custom-designed mission applications and the integration of DII COE components. Consideration of the network environment is also essential to provide the context for distributed application design and usage in the DoD context.

The division of computing tasks is a design issue for customized mission applications and affects the stability of the COE components. This section provides information for planning purposes but does not address the programming details covered in Section 4.5 of this manual (Distributed Computing Environment).

A well-planned integrated application is network infrastructure dependent and provides an environment that may grow or shrink dynamically or progressively, from local to regional and global interactions, as the DoD mission requirements change. The progression from simple (local) to universal (global) use increases the complexity of applications and system deployment, operations, and support incrementally. However, the basic distributed applications, as designed, should not change in the process.

The client/server model is well suited for an increasingly universal network environment that requires global connectivity to serve the Warfighter. Mission applications are being designed to meet these operational and internetworking needs. The benefits of the client/server model capabilities are:

- Peer Level Computing
- Multiple LAN Bridging
- LAN to WAN Interconnection
- Enterprise-Wide Connectivity
- Global Mission Internetworking
- Universal Systems Interoperability.

It is important to recognize that each aspect of networked distributed computing requires distinct application-level services. The division of tasks between the client and the server is an essential design issue for minimizing network traffic. The distribution of tasks between the client and the server or server to server determines the extent of the application's reach and interoperability across the network.

2.5.2.2 Application/Data Separation

Achieving a high level of application portability and component reuse requires development methods and tools that guarantee a level of independence. The separation of application processing from the semantics of data is a primary consideration to develop programs that are independent of the vendor-specific database or data warehouse in use. Similarly the DII COE infrastructure (as defined by TAFIM and related OSE standards) requires the independence of application components from operating system services. Separation of those services that are native to the computer in use versus standards-defined POSIX services and interfaces is the key. This separation assures the means to create portable and reusable applications components as well as common functional modules (or subroutine calls) that interact with mission applications to provide the highest degree of cross-application integration.

2.5.2.3 Design for Reuse and Porting

The DII COE application development services support reuse and portability of mission applications. These services use methods and infrastructure as defined by TAFIM and related OSE standards. The COE provides the means to create portable and reusable applications components (and common functional modules) that interact with other applications residing in different computers regardless of make and model.

The most important aspect of developing customized portable applications is to scope the tasks and clearly define the objectives of the client as well as the server. A distributed application that uses OSE application services must use APIs. APIs are standards-based and provide appropriate programming language bindings (i.e., Ada and C).

Specialized or custom distributed application capability requires an in-depth understanding of the business process and operational requirements. Moreover, the designer must understand the functional limitations of the environment that is being created.

Understanding the methods and automating the business/operations processes are not sufficient to develop an optimum distributed processing environment. Sometimes, a redesign of the business process is necessary to maximize the benefit of distributed computing. Another aspect of designing and developing a distributed application is understanding the current capabilities of the network, as well as, planned future enhancements. This is important because applications need to scale and transition to future technologies and mission realities without excessive constraints. Some considerations and elements are as follows:

- Understanding of mission and processes
- Responsiveness of the network to the enterprise needs
- Seamless distributed processing across the network
- “Reachability” -- local, area, regional, national, international.

2.5.3 Environment Stability

In designing and developing distributed applications, several aspects of the systems environment must be considered to assure a maximum life-cycle for the data (information), the applications (software) and the expertise (human). The issues listed below must be considered when building a durable and robust environment for the design of portable mission applications. At a high level, these issues are:

Platform - Hardware/Software -- Operating Environment

- Types of Platforms and Operating Systems
- Updates - Maintenance Releases
- Enhancement Releases
- Expandability

Standards for OSE Distributed Computing

- Applications Development Tools
- Data Repository and Databases
- Application Program Interfaces
- Network Applications Services
- Network Transport Services

Vendors/Suppliers' Products

- Open Architecture
- Cost and Performance
- Compliance to Standards
- Availability of Utilities and Tools
- Use of Middleware and Groupware
- Product Quality, Stability and Usability
- Development Expertise, Support and Training

Enterprise Environment

- Mission Urgency
- Use of Technology
- Culture and Policies
- Operations and Processes
- Controls and Applied Procedures
- Work and Business Process Flow Knowledge

Portability and Interoperability

- User's and Programmer's Expertise
- Defense Wide Applications Requirements
- Information Availability Life Cycle
- Joint Task Forces, Partnership Exchanges and Collaborations

Mission Considerations

- Management Support
- Strategies and Tactics
- Assets and Process Modeling of Resources
- Technical Support and Training
- Information Infrastructure Support

2.5.4 Division of Computing Tasks

To build distributed applications in a client/server environment, it is necessary to determine the division of the computing tasks required and where these tasks will reside (i.e., the client or the server). The fundamental concept is to analyze the tasks required for the application and logically unbundle the distinct processes that comprise the entire application. For example, a traditional application may have the following components: user interface, application logic, data manipulation or management, data repository (disk device handler), and various data output media. The application logic may be further subdivided into data validation functions and computational logic. These functions may reside individually in separate processors tied together by a communications facility. The communication facility may be a LAN, a WAN, or both. The communications facility is provided by a suite of protocols that can span across network nodes that reside within LANs and WANs.

Furthermore, distributed applications may be structured like a tree where each branch provides different application logic, the leaves are services or instances of processes or data elements, and the trunk and roots are the base infrastructure support. Other distributed applications may be simple, using the basic requester and responder relationship where each request triggers a specific response.

Figure 2-4 shows the division of tasks between clients (requesters) and servers (responders) or between various servers. This division can be viewed as sequences of requests and responses. The client task can exist independently on its own (self-sufficient), but may require the server to complete a distributed task. The various models of task division may cross other application processes and be shared by cooperating systems. The paradigm of using computers that serve both client and server roles gives way to developing specialized applications that specifically optimize the use of the processor's capabilities and resources. Hence, the demarcation line of the division of labor (tasks) may vary between different application components, but must remain consistent within a specific application that is distributed across the network.

2.5.5 OSE Enabled COE Applications

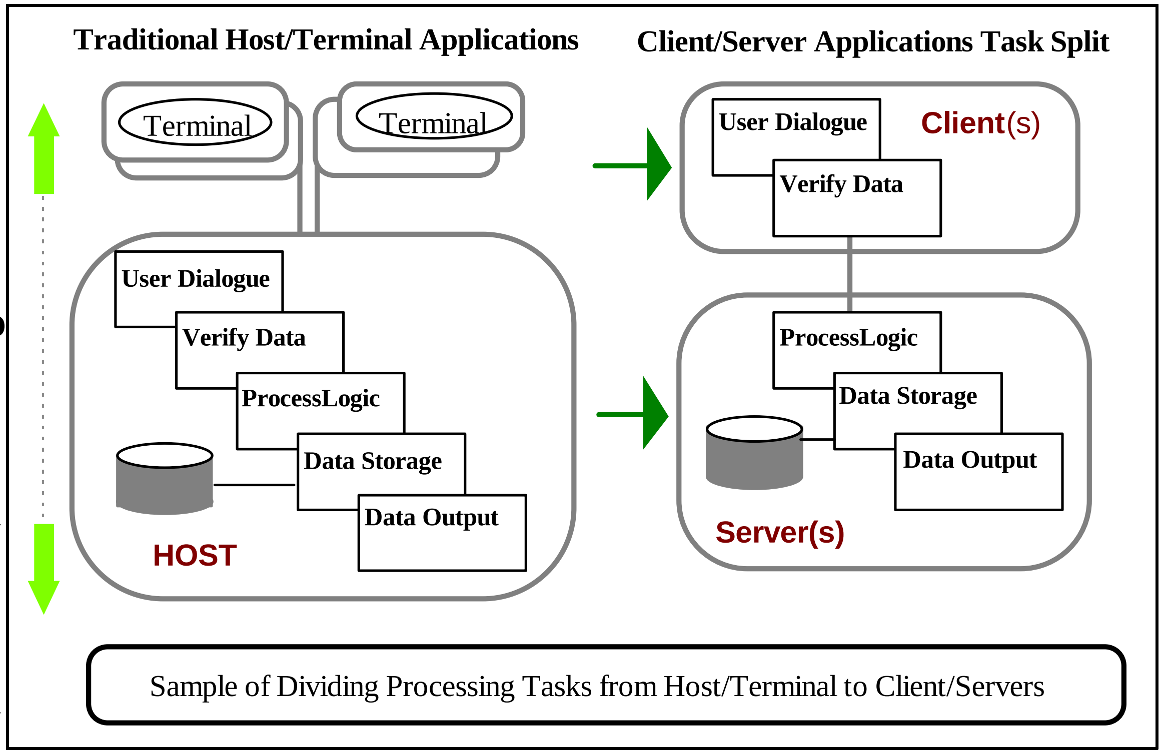


Figure 2-4. Division of Client/Server Tasks

The TAFIM is based on OSE (IEEE P1003.0) and is the foundation of the DII COE. OSE application services and protocols enable various mission and business application processes to interwork over diverse networks while residing on heterogeneous platforms. The advantage of OSE and DII COE concepts is that the applications themselves provide specific system-level services that may be used selectively with or without operator intervention. The user may be a human interaction via a user-interface facility (i.e., GUI), another specialized mission, or an operational application that needs the OSE services.

For instance, the use of X.400 (Message Handling System) provides the transport mechanism for compound documents or multimedia binary objects using a store-and-forward mechanism. This E-mail mechanism works like the postal service to relieve the sender of the need to monitor the delivery of the message and its objects to its destination. Similarly, a mission application may choose to send an electronic message (mail item) automatically in the form of a transaction notification. This is accomplished by inserting a formatted or free-form message directed to another process or entity with assurance of delivery and arrival status of the messages sent.

Mission applications may use some or all of the service facilities of the COE suite. Besides the use of typical E-mail transport, COE application facilities include: network utility-type services for Distributed Directory, File Transfer, Remote Data Access, Transaction Processing, and specialized application layer services such as Remote Procedure Calls (RPC), Open Software Foundation/Distributed Computing Environment (OSF/DCE), Manufacturing Message Services or Information Storage and Retrieval that provide COE enabled applications across DII.

2.5.6 Distributed COE Applications

The COE application services were intended to be used in OSE as generalized utilities to serve various mission and enterprise applications needs. This concept of using OSE applications as generic application utilities has matured and can be used by programmers today by way of functional calls using APIs.

An overwhelming majority of programmer-written applications require only basic communication services: 1) open connection, 2) send and receive messages 3) close connection, and 4) handle errors. In a connection-oriented mode the outcome of any communication service request is always acknowledged.

All network services and communications based application protocols provide utilities and interface services that can be used to support the development of distributed mission applications. The application developer can write applications that take full advantage of these services thus enhancing the customer's application development environment to meet their specific mission requirements. By using these generic network application services, programmer-written applications can provide a strategic advantage to DoD Agencies/Services Information Resource Managers (IRM) in overcoming the backlog of software development.

The distribution of computing tasks as "peer-level" computing services allows software on multiple, independent, heterogeneous processors to cooperate in providing effective delivery of application services. COE provides the foundation services for building the DII and the means to create OSE compliant applications. DII COE provides the ability to work cooperatively across multiple, heterogeneous systems and platforms to accomplish the DoD mission.

2.6 Integration and Runtime Specification

The *DII COE Integration and Runtime Specification (I&RTS)* delineates technical requirements for using the DII COE to build and integrate systems. The *I&RTS* document provides additional details on many of the topics covered in this manual and in the Programmers Reference Manual. The *I&RTS* provides detailed implementation details that describe, from a software development perspective, the following:

- the COE approach to software reuse,
- the COE runtime execution environment,
- the definition and requirements for achieving COE compliance,
- the process for automated software integration, and
- the process for electronically submitting/retrieving software components to/from the COE software repository.

The initial development of the DII COE is primarily in two systems: the GCCS and the GCSS. Both systems use the same infrastructure and integration approach, and the same COE components for functions that are common.

GCCS is a C4I system with two main objectives: the near-term replacement of the World-Wide Military Command and Control System (WWMCCS) and the implementation of the C4I for the Warrior. Functionally, as a C4I for the Warrior system, GCCS includes multiple workstations cooperating in a distributed LAN/WAN environment. Key features include 'push/pull' data exchange, data processing, sensor fusion, dynamic situation display, analysis and briefing support, and maintenance of a common tactical picture among distributed GCCS sites. GCCS is already fielded at a number of operational CINCs.

The present COE is being expanded to include global data management and workflow management for GCSS logistics applications. It will expand further as more functional areas desire to employ its services in areas such as Electronic Commerce/Electronic Data Interchange (EC/EDI), transportation, base support, personnel, health affairs, and finance. The COE is described more completely in Section 4.0 COE Components. This document also describes technical information required to properly access and extend software contained within the COE.

2.6.1 System Management

The COE provides templates for 'account group segments' to be used in establishing individual login accounts. Account groups contain template files for defining what COE processes to launch at login time, what functions are to be made available to operators, and preferences such as color selections for window borders. Account groups can also be used to perform a first level division of operators according to how they will use the system. This technique is used in the COE to identify at least five distinct account groups:

- Privileged Operator (e.g., root),
- System Administrator,
- Security Administrator,
- Database Administrator, and
- Non-Privileged Operator.

Additional details on the functions available to each group are in the *I&RTS* document; however, command-line access for privileged operator accounts is expressly prohibited without prior approval from the DISA Chief Engineer. Within an account group, subsets of the available functionality can be created. These subsets are called *profiles*. An operator may participate in multiple account groups with multiple profiles, and can switch from one profile to another without the need to log-out and log-in again. Other account groups may exist for specialized system requirements, such as providing a character-based interface, but all account groups follow the same rules.

2.6.2 Variants

It is neither desirable nor feasible to install all software and all data on every workstation. Instead, only a subset of the segments are installed on any particular workstation. The kernel COE is required to be on each workstation, but additional segments are dependent upon how the workstation will be used. The exact configuration of segments installed is called a *variant*. A variant is simply a collection of segments that are grouped together for installation convenience. For example, it is more convenient for an operator to indicate that a workstation is to act as a database server (a variant), or used as an intelligence analyst workstation (another variant) than to manually select all of the segments that need to be installed. The COE is designed so that a site may install predefined variants, or customize the installation to suit site specific requirements.

There are several types of variants which have been identified:

- System variant is a collection of segments required to provide functions for a very specific problem domain (e.g., GCCS for C4I, GCSS for mission support).
- **Site variant** is a subset of segments from the system variant which are required to meet site specific mission objectives (e.g., JTF, USACOM, USPACFLT, TRANSCOM).
- **Mission area variant** and *workstation variants* support workstations grouped together by mission area into physical spaces, rooms, or other logical arrangements.

2.6.3 Development Process

The *I&RTS* describes the development process in detail. A powerful feature of the overall development process is the concept of automated integration. This means that automated tools are used to combine and load segments, make environmental modifications requested by segments, make newly loaded segments available to authorized users, and identify places where segments conflict with each other. Traditional system level integration then becomes primarily a task of loading and testing segments. Traditional integration tasks are pushed as far down to the developer level as possible.

Prior to submitting a component to DISA, a developer must:

- package the component as a segment,
- demonstrate COE compliance through tools and checklists,
- test the segment in isolation with the COE,
- provide required segment documentation, and
- demonstrate the segment operating within the COE.

The Software Support Activity (SSA) enters the segment into the on-line library for configuration management purposes and confirms COE compliance by running the same suite of tools as the developer. The SSA then tests interaction between segments and the impact on performance, memory utilization, etc. Since segments typically can only interact through the COE, the task is greatly simplified and the need for human intervention in the process is minimized.

An automated integration approach is a practical necessity. Not only are segments contributed by different services and agencies, but individual segments are also created by a large body of different developers. Traditional integration approaches rapidly break down with the need to communicate to such a large number of people while the costs incurred to resolve inter-module conflicts at system integration time become prohibitive.

The COE approach is designed to be non-intrusive; it places minimal constraints on how developers build, test, and manage software development. It concentrates on the end product and how it will integrate in with the overall system. This approach provides the flexibility to allow developers to conform to their internal development process requirements. However, developers are expected to use proven software engineering practices and development tools to ensure robust products. Developers are free to establish a software development environment that is best suited to their project. The COE requires only that deliveries are packaged as segments, that segments are validated before submission, and segments are tested in the COE prior to submission. Chapter 6 in the *I&RTS* provides detailed lists of requirements and suggestions regarding coding practices, development file structures, and libraries for scripts and files.

SECTION 3.0 COE CORE SERVICES

3.1 COE Kernel

The COE is emerging as a standard made up of two areas. One of the areas is required for every application is referred to as COE kernel. The COE *kernel* is the mandatory minimal set of software required on every workstation regardless of which workstation will be used. The kernel COE components are the shaded boxes in Figure 2-2 DII COE Model. A kernel COE will always contain the following:

- Operating System
- Windowing Services
- External Environment Interfaces.

It will also include the following features:

- Basic System Administration function
- Basic Security Administration function
- Executive Manager function (e.g., a desktop GUI)
- Template for creating privileged operator logging accounts
- Template for creating non-privileged operator logging accounts.

The System Administration segment is required because it contains the software necessary to load all other segments. The Security Administration segment is required because it enforces the system security policy. The Executive Manager is required because it is the interface through which an operator issues commands to the system. The Executive Manager is an icon and menu driven desktop interface, not a command line interface. The templates included in the kernel COE describe the basic runtime environment context that an operator inherits when he/she logs into the system. This login defines which processes run in the background, which environment variables are defined, etc. The kernel COE assures that every workstation in the system operates and behaves in a consistent manner, and that every workstation begins with the same basic components within the systems environment.

The COE provides the flexibility to extend the base environment by adding segments as required.

3.2 Runtime Environment

This section describes the software configuration for the COE runtime environment. All software and data, except for low-level components of the bootstrap COE, are packaged as segments. A **segment** is a collection of one or more CSCIs most conveniently managed as a unit. Segments are constructed to keep related CSCIs together so that functionality may be easily included or excluded.

Segment installation is accomplished in a disciplined way through instructions contained in files provided with each segment. These files are called *segment descriptor files* and are contained in a special subdirectory, `SegDescrip`, called the *segment descriptor subdirectory*. Installation tools process the segment descriptor files to create a carefully controlled approach to adding or deleting system segments. The format and contents of the segment descriptor files are the central topic of this section.

The segment concept and the strict rules which govern the COE and runtime environment provide several benefits:

- Segment developers are decoupled and isolated from one another.
- Extensions to the COE are coordinated through automated software tools.
- Compliance verification and installation can be automated.
- Mission application segments are isolated from the COE.

Principles contained in this section are fundamental to the successful operation of the COE, and COE compliance is largely measured by this chapter. Developers are required to adhere to the procedures described herein to ensure that segments can be installed and removed correctly, and that segments do not adversely impact one another. For additional information on the Runtime Environment, see the *DII COE I&RTS*, Section 5.

3.2.1 Disk Directory Layout

A standardized disk directory structure for all segments is required to prevent two segments from overwriting the same file, creating two different files with the same name, or similar issues that cause integration problems. Unfortunately, such problems are often not discovered until the system is operational in the field.

In the COE approach, each segment is assigned its own unique, self contained subdirectory. A segment is not allowed to directly modify any file or resource it doesn't 'own' (i.e., outside its assigned directory). Files outside a segment's directory are called *community files*. COE tools coordinate modification of all community files at installation time, while APIs to the segments which own the data are used at runtime.

The three main types of subdirectories are:

- Segment Subdirectories

  COE compliance mandates specific subdirectories and files underneath a segment directory. The precise subdirectories and files required depend upon the segment type. Runtime subdirectories normally required are `data, bin, scripts,` and `SegDescrip`.

- Users Subdirectories

  The COE establishes individual operator login accounts and provides a separate subdirectory on the disk for storing operator specific data items. This structure is created and managed automatically as accounts are added or deleted by the Security Administrator software.

- Developer Subdirectories

  Software for the runtime environment is obtained by loading the desired mission application segments and the required COE components. But the development environment is provided separately as a Developer's Toolkit because it is not delivered to, nor required at, an operational site. The Developer's Toolkit includes object code libraries, header files which define the public APIs, and various tools.

3.2.2 Segment Prefixes and Reserved Symbols

Each segment is assigned a unique subdirectory underneath /h called the *segment's assigned directory*. The assigned directory serves to uniquely identify each segment, but it is too cumbersome for use in naming public symbols. Therefore, each segment is also assigned a 1-6 character alphanumeric string called the *segment prefix*. The segment prefix is used for naming environment variables and for public APIs and public libraries, where naming conflicts with other segments must be avoided. All segments shall preface their environment variables with `segprefix_` where `segprefix` is the segment's assigned prefix. For example, the Security Administrator account group segment is assigned the segment prefix `SSO`. All environment variables for this segment are therefore prefaced with this prefix (i.e., the string "`SSO_`").

The segment prefix is also used to uniquely name executables. All COE component segments shall use the segment prefix to name executables, and it is strongly recommended that all segments follow the same convention. This approach simplifies the task of determining the files that go with each segment and reduces the probability of naming conflicts.

3.2.3 Segment Types and Attributes

This section describes segment types and attributes. Segments are the cornerstone of the COE approach. Developers have considerable freedom in building segments. For more detailed information, see the *DII COE I&RTS*.

3.2.3.1 COTS Segments

The COTS segment type is used to describe the installation of COTS products. If a COTS product can be structured as a software segment, it should be. However, this is usually not possible because - where COTS products will be loaded, what environment extensions are required, etc. - are often vendor specific.

3.2.3.2 Data Segments

Data files are most often created explicitly at runtime by a segment, or loaded as part of the segment itself. However, the ability to load data as a separate segment is useful when there is classified data, optional data, or data that may not be released to all communities. The COE supports five categories of data grouped according to data scope, how the data is accessed, and where the data is located: Global, Database, Local, Segment, and Operator.

3.2.3.3 Database Segment

A database segment contains everything that is to be installed on the database server under the management of the DBMS and the ownership of the DBA. It contains the component database and any utilities provided by the developers for the DBA's use in installing and filling that particular database. Database segments may only be installed on a database server.

3.2.3.4 Account Group Segments

An account group segment is a template for establishing a basic runtime environment context that other segments may extend in a controlled fashion. An account group segment determines:

- the processes to launch,
- the order in which to launch processes, and
- the required environment script files.

3.2.3.5 Software Segments

Software segments add functionality to one or more account groups. The account group(s) to which the software segment applies is called the *affected account group(s)*.

3.2.3.6 Patch Segments

The COE supports the ability to install field patches on an installed software base. A patch segment means the replacement of a collection of one or more individual files, including those of the operating system. It does **not** refer to overwriting a portion of a file, as is sometimes done to patch a section of binary code.

3.2.3.7 Aggregate Attribute

It is sometimes convenient for a collection of segments to be treated as an indivisible unit. The aggregate attribute provides this capability, and the collection of segments are called an *aggregate segment*. One, and only one, segment is designated as the *parent* segment and the remaining segments are designated as *children*. Parent and child segments are designated as members of an aggregate in the `SegType` descriptor file. The child segment must list its parent segment, while the parent segment must list each child in the aggregate.

3.2.3.8 COE Component Attribute

Authorized segments may specify the attribute of being a COE component segment. COE component segments are similar to aggregate segments in that one segment serves the role of a parent segment and all others are children to that parent. The parent segment is similar to an account group segment which is affected by a collection of child component segments. However, there are important differences between COE component segments and aggregate segments, and between the parent COE component segment and account groups.

3.2.3.9 Segment Dependencies

Segments specify dependencies upon one another through the `Requires` descriptor. However, the COE does not allow circular dependencies. That is, a situation such as SegÊA depends upon SegÊB, SegÊB depends upon SegÊC, and SegÊC depends upon SegÊA is strictly forbidden.

Components of an aggregate may have dependencies upon other components within the same aggregate. But since components of an aggregate are always loaded together as a unit, this does not pose a problem. Components of an aggregate must **not** specify dependencies upon one another in the `Requires` file, even if this is indeed the case. Likewise, the parent segment must *not* specify a dependency on children within the aggregate.

3.2.4 Segment Descriptors

This section details the contents of the segment descriptor files. These files are the key to providing seamless and coordinated systems integration across all segments. Adherence to the format described here is required for all segments to ensure COE compliance. This enables automatic verification and installation of segments.

3.2.5 Segment Installation

Segment installation requires some form of electronic media (tape, disk, etc.) that contains the segments, and that has a table of contents which lists the available segments. `MakeInstall` is the tool which creates this electronic media. However, it is important to identify the operations (e.g., compression) performed on segments, and the sequence in which these operations are performed.

Installation requires reading the table of contents created by `MakeInstall`, selecting the segments or variants to install, and then copying the segments to disk. Segments may actively participate in the installation process through `PostInstall`, `PreInstall`, and `DEINSTALL` scripts. For detailed information on the `MakeInstall` tool, and the installation sequence see the *DII COE I&RTS*.

3.2.6 Extending the COE

Most properly designed segments will not require any extensions to the COE, except for the need to add icons and menu items. This subsection lists some of the more commonly required extensions. For details on the commonly required extensions and techniques for addressing less frequently encountered extensions see the *DII COE I&RTS*.

- Adding Menu Items to the Desktop
- Adding Icons to the Desktop
- Modifying Window Behavior

- Using Environment Extension Files
- Using Community Files
- Defining Background Processes
- Reserving Disk Space
- Using Temporary Disk Space
- Defining Sockets
- Adding and Deleting User Accounts
- Adding Network Host Table Entries
- Registering Servers
- Modifying Network Configuration Files
- Establishing Network File Server (NFS) Mount Points
- License Manager
- Shared Libraries
- Character Based Interface
- Remote versus Local Segment Execution
- Color Table Usage.

3.2.7 Database Considerations

Typically, COE-based systems are heavily database oriented. Database considerations are therefore of paramount importance in properly architecting and building a system. For more detailed technical information on properly designing databases and database applications see Section 5.9 of the *DII COE I&RTS*.

3.2.7.1 Database Segmentation Principles

A COE database server is provided by a COTS DBMS product. It is used in common by multiple applications. It is a services segment and part of the COE. However, different sites need varying combinations of applications and databases. As a result, databases cannot be included in the DBMS segment. Instead, these component databases are provided in a database segment established by the developer. The applications themselves are in a software segment, also established by the developer, but separate from the database segment. If the data fill for the database contains classified data, that data fill must be in a separate data segment associated with the database segment.

Database Segments

The DBMS is provided as one or more COTS segments. These segments contain the DBMS executables, the core database configuration, database administration utilities, DBMS network executables, and development tools provided by the DBMS vendor. Databases are provided as database segments. These segments contain the executables and scripts to create a database, and tools to load data into the database.

Database Segmentation Responsibilities

Three groups are involved in the implementation of database segments: DISA, the application developers, and the sites' database administrators. The developers and DISA work together to field databases and associated services for the DBAs to maintain. DISA provides the DBMS as part of the COE. Developers provide the component databases. Sites manage access and maintain the data. Users interact with the databases through mission applications and may, depending on the application, be responsible for the modification and maintenance of data in the databases.

DBMS Tuning and Customization

The core DBMS instance is configured and tuned by DISA based on the combined requirements of all developers' databases. This allows the DBMS Server Segments to be reasonably independent of particular hardware configurations and ignorant of specific application sets.

The final tuning of the DBMS cannot be accomplished until a complete configuration is built and it has an operational load. Developers should provide information for the tuning process, but should not make their applications dependent on particular tuning parameters. Where a non-standard parameter is required for operations, developers must provide that information to DISA so the DBMS services segment can be modified accordingly.

Developers shall not modify the core DBMS instance's configuration. Extensions or modifications of that configuration require the specific approval of the DISA Chief Engineer.

If developers modify any of the executable tools (e.g., add User Exits to Oracle*Forms), then the modified version of the tool does not reside with the core database services, but becomes a part of the application's client segment. This prevents conflicts among different modified versions of a core function. The maintenance of that modified tool also becomes the responsibility of the developers.

3.2.7.2 Database Inter-Segment Dependencies

A key objective of the segmentation approach is to limit the interdependencies among segments. Ideally, database segments should not create data objects in any other schema or own data objects that are dependent on other schemas. However, one purpose in having a Database Server is to limit data redundancy and provide common shared data sets. This means that there will usually be some dependencies among the databases in the federation. For additional information, see the *DII COE I&RTS.*

3.3 COE Tools

One of the strongest features of COE is its wide collection of tools. These tools can be subdivided into two user categories; 1) the runtime tools which are part of the delivery code, and 2) development tools which are only for developers use. These tools can further be subdivided into five different types of toolkits as follows:

2. COE API reference guide toolkit
2. COE Developer's toolkit (HP and Solaris)
2. COE Printer API reference guide toolkit (HP and Solaris)
2. COE Printer user profiles API toolkit
2. COE database toolkit.

See Appendix F (DII COE Description of Tools).

3.4 API Overview

Application Program Interfaces (API) provide applications services from the application platform components for programmers to use in their application development. It provides a level of portability and independence from any underlying platform services.

3.4.1 Introduction

This section describes the API model and functional considerations for the design and coding of DII applications. To facilitate DII portability and re-use across platforms requires the ability to port applications, data, and expertise across systems. Mission application portability requires the use of standardized APIs. This section discusses how programmers use the DII COE APIs. It also covers the availability of DII COE API development tools and related procedures.

APIs aid developers who with minimal difficulty need to take advantage of services provided in the DII COE components. A primary requirement of APIs is that they be stable and provide the specific services in consistent manner. That is, the interface remains constant even if the details of the underlying implementation of the service vary. If APIs do not remain constant, then applications which rely on them will cease to function without significant code modification. To manage change in a complex system, the changes must be isolated to as small an area as possible. The use of APIs makes it possible to minimize the impact of underlying changes and make reuse feasible.

It is absolutely vital that the applications access services only through common APIs. The use of private (i.e., undocumented or not COE approved) APIs make component upgrades difficult and cause mission applications which use those private APIs to cease to function correctly.

The Developer's Toolkit of the DII COE includes APIs to access services to:

- Help in the development of new applications.
- Help developers provide information to users and solicit input from them. These are intended to assist when new applications (segments) which have a broader applicability for developers are installed or removed from a system.
- Print data.
- Provide mapping functions.
- Use the Distributed Computing Environment (DCE) for implementing distributed applications with a client/server model.

The APIs are documented in *Programmers' Reference Manual* and various DII COE documents which are part of the COE Toolkit.

APIs for the Common Desktop Environment (CDE), Oracle, and Sybase are not included in the toolkit, but are provided and documented by their respective developers.

The Developer's Toolkit is not provided as a DII COE-compliant segment but as a `tar` file. Details on how to load the tools onto a DII COE-based system are provided in the documentation that comes with the toolkit (*Installation Instructions for the Developers Toolkit).*

3.4.2 Role of APIs in Application Development

The use of API tools makes application development more challenging and efficient. The access to systems and network services by DII applications segments or other mission applications requires the use of program service calls using APIs. In application development, developers use APIs to request the services of the common systems components such as directory, file transfer, E-mail, Remote Database Access, etc. The service calls provide specific application service entities (ASE) that are accessible by the appropriate application program linkages to the language binding services. The APIs are the means of accessing the service elements of all DII network and platform services.

3.4.3 Technical Requirements

The technical requirements for the interfaces between services and user developed modules or the operating services are the building blocks that can be listed in a functional specification or profile. The profile must be well documented to provide consistent and well-defined interchange between application modules as well as to the external and internal environments. Each interface requirement specifies and describes an interface between two modules within the profile or between the module and the external entity. The external entities are described in terms of their interaction with the profile. A protocol may provide connectivity to a different system that is not detailed, or an API may be used by other compliant applications.

3.4.3.1 Interface Specifications for Portability

There are primarily two classes of interfaces important to the development of application portability. They are APIs and Platform External Environment Interface (EEI), as referenced by *TAFIM* Volume 2. Specifically, the API is the internal interface between application software and the systems software platform. The platform EEI is the external interface between application software platform and the users or the wares of the external world (e.g., display unit, keyboard, mouse, wires, and various peripherals).

There are additional distinctions for interface classes that are associated with:

- Look and feel interface between human and machine
- Formats for exchange and process of data
- Module interfaces such as:

- Source program interfaces (language bindings)
- Binary program interfaces (binary bindings)
- Protocol mechanisms for information exchange between modules (i.e.; communications and revisable formatted information exchange).

The most commonly understood and applicable interface for DII COE for Application Portability Profile (APP) is the internal software interface known as APIs. APIs provide a service access method for transferring data and invoking (activating) functional services within a system or across components, segments, and modules or layers of logical and physical system software. See Figure 3-1 for a view of the API boundary in an applications context.

A common definition of an API is:

> An API specifies the mapping between program syntax and the features of a specific service, and thereby provides access to that service from applications written in a particular programming language, when the application is bound to the service by the language implementation. (IEEE - PASC POSIX)

An alternate short definition is; “an interface is a boundary between two entities for purposes of processing data.” For example, an API is the interface between application software and the underlying application software platform services. The platform is a set of organized resources on which the application will run, and which provides all the services required by the application.

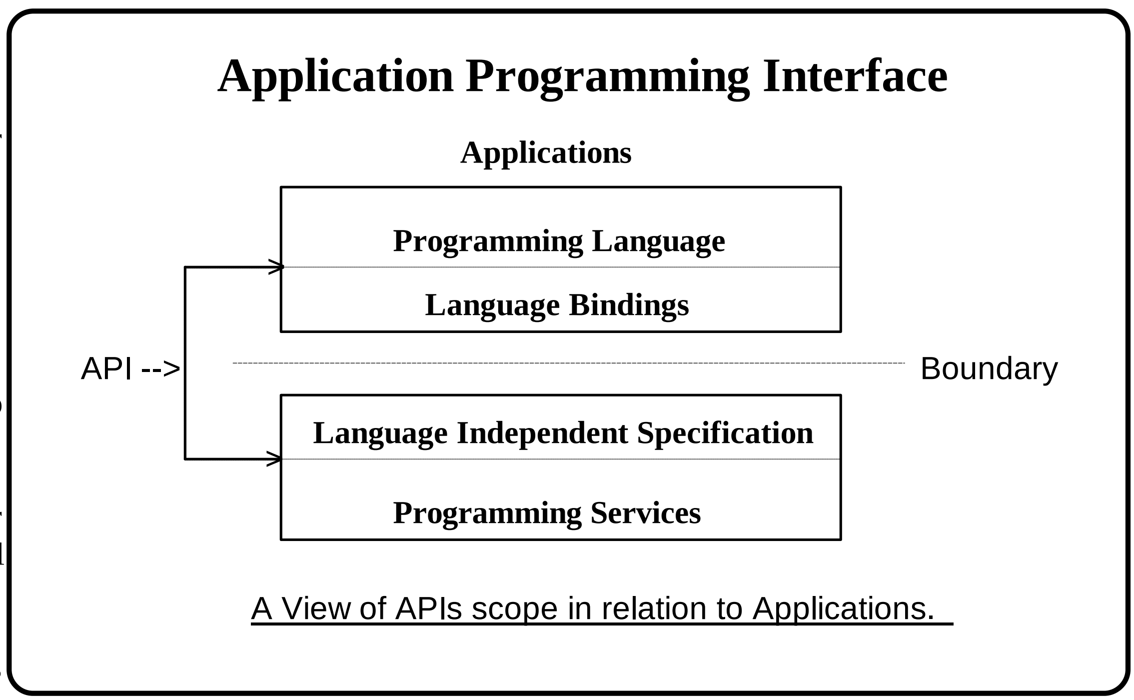


Figure 3-1. API Interface Boundaries Between Application and Services

All systems interfaces are classified according to their level of openness. This technical classification defines different levels of programmatic details within the interface. The technical requirement is documented with explanatory text and rationale that points to the original user requirements. The APP is further discussed in Section 3.4.5.

3.4.3.2 API Requirements for DII COE

The current state of API standards for DII are on a fast track and developing rapidly. In North America, the IEEE PASC is the formal standards body responsible for API standards. IEEE is the author of POSIX interface for operating systems services and has completed several specifications for networking application service interfaces. IEEE standards are submitted to the International Standards Organization (ISO/IEC JTC1 SC21 and SC22) for international consensus building and acceptance.

To understand the use of APIs, it is useful to have a contextual reference point. Figure 3-2 provides an example of various networking interfaces developed for the OSI reference model. Figure 3-2 provides a view of existing APIs from IEEE, X/Open and other consortia for use in the OSI application layer services. Additionally, a specific API provides access to the transport layer services. The transport layer interface (TLI [API]) was specified to access both OSI and TCP/IP protocols. It hides the complexities of the transport protocol and related requirements while allowing programmers to access both OSI TP and TCP/UDP using the X/Open specification of XTI and the UNIX “socket” interface.

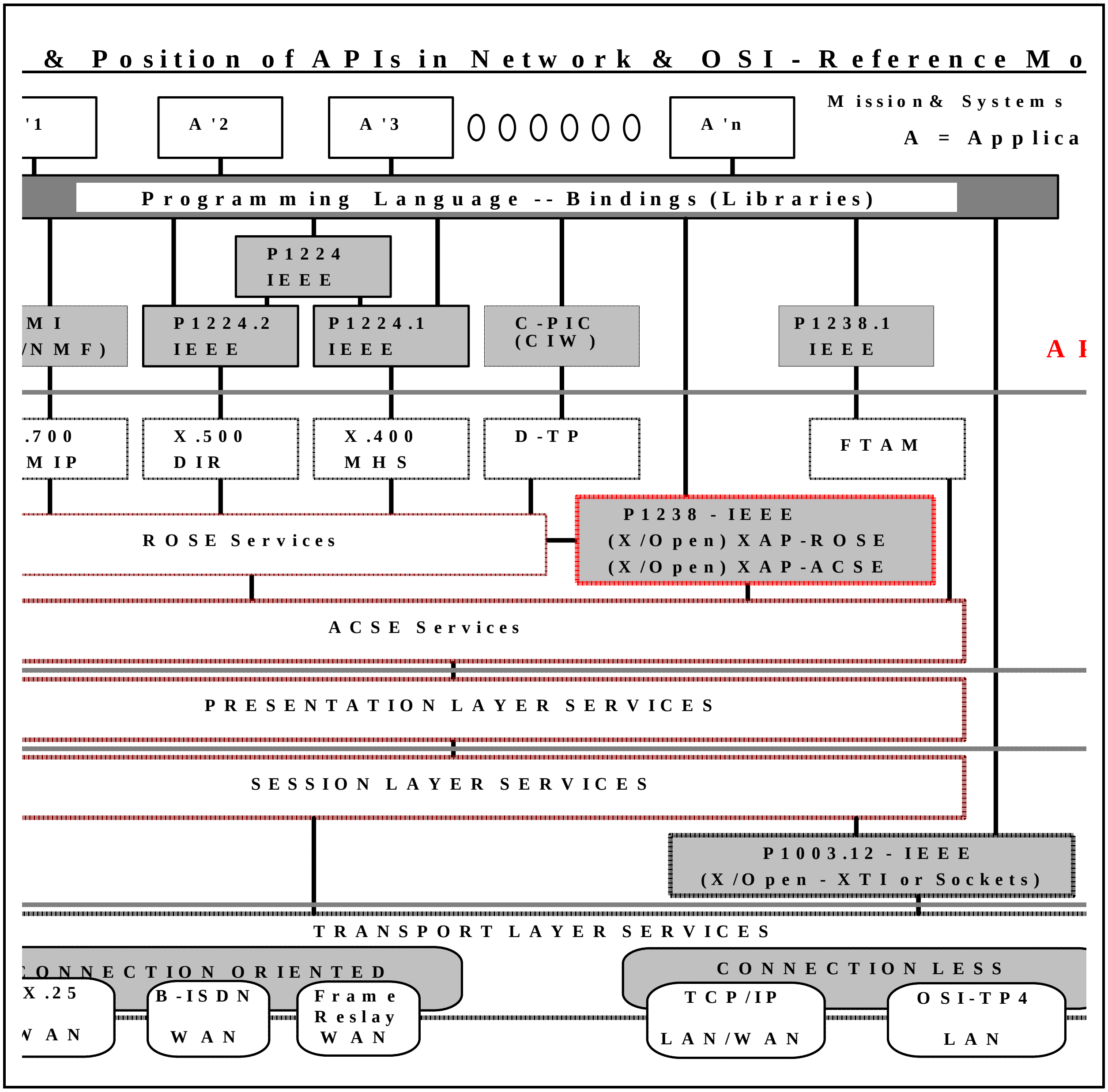


Figure 3-2. Sample View of API Placement in OSI-RM

3.4.4 Context of APIs for DII Profiles in an OSE Model

The elements of OSE based DII profiling are:

- model and framework
- user requirements
- Technical Functional Specification
- OSE Specification Profile.

Various standards bodies are developing requirements and related initiatives including:

UR - User Requirements process

TF - Technical Functional description

SP - OSE Specifications Profile(s)

URM - User Requirements Model

APP - Application Portability Profile

FSP - Functional Standards Profile

3.4.5 Application Portability Profile

The Application Portability Profile (APP) created by National Institute of Standards and Technology (NIST) is an Open Systems Environment (OSE) profile developed for use by the US. Government. It consists of User Requirements and Specification Profile(s). The APP is primarily a procurement document addressing functional needs expressed and based on users' requirements and various standards or technical specifications.

The technical Specification Profile (SP) consists of sets of standard systems services or user developed components. A Specification Profile, once standardized, is known as a Functional Standards Profile (FSP) or OSE Profile. Once an FSP is approved for standardization it is registered and submitted to the International Standards Organization (ISO/IEC JTC1 SGFS) for developing a consensus in the technical committee, and upon approval is labeled an International Standard Profile (ISP).

Figure 3-3 is an enhanced model of APP and contains the same elements as the *TAFIM*. The OSE APP is based on support for the IEEE POSIX P1003.0 *OSE Guide for APIs* and contributes to standards development for OSE Profiles for Networking, Data Interchange, Repository, Operating Systems, Graphics, Application Development, User Interfaces, and Management and Security.

The OSE APP (see Figure 3-3) is an organized group of generic Information Technology Standards specified for purposes of cross referencing and mapping to the Technology Integration templates and Functional Standards Profiles. According to OSE profiling requirements, the profiles may reference non-standard specifications (industry standards) which may be based on other Functional Standards Profiles (or templates). The OSE APP provides a model to create the DII COE "plug and play" concept for segments, components and other devices which are compliant to the *TAFIM*.

3.4.6 Application Validation and Conformance Requirements

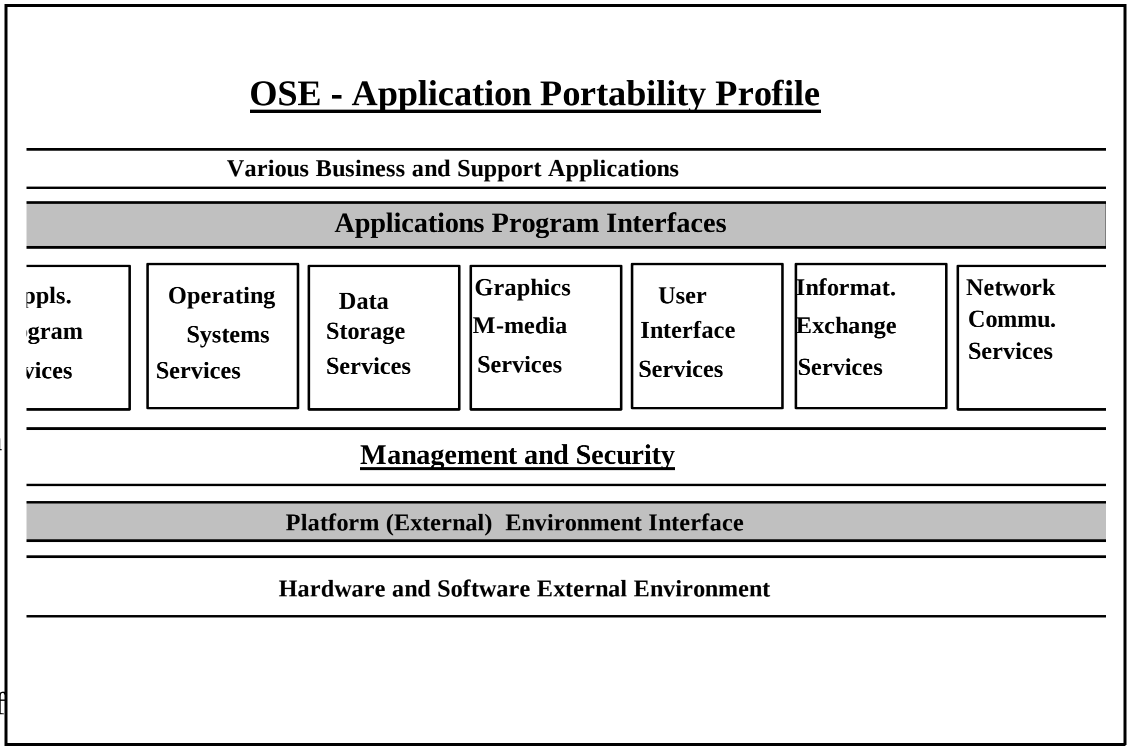


Figure 3-3. NIST Application Portability Profile (OSE/APP)

Conformance for user created OSE Profiles may be determined in various ways but currently there are no provisions for testing and certification. When formal DII OSE Profiles and accompanying documentation are created, an established process for DII documentation, verification, conformance, and test may be established by the appropriate DoD agency.

For DII COE Segments and Components, there are various requirements for compliance to the conventions for re-use and portability. These are listed in more detail in Section 5.1.

SECTION 4.0 COE - COMPONENTS

4.1 COE Components

Section 2 discussed in general the concept of families of segments which can be grouped together to form the components of the COE. This section addresses the various components contained in Version 2.0. This section provides application development details on:

- Databases - Interaction with COTS database products
- Common Desktop Environment - Functionality provided by CDE, style guide overview
- Security - Access control mechanisms and classification considerations
- Distributed Computing Environment - Distributing workload in a client - server environment
- Communications - Networking services including multi-level security
- Other Auxiliary Services - General commercial Internet applications.

4.2 DII COE - Building the Database

4.2.1 COE Database Concepts

The DII COE uses database servers within the client/server architecture model to provide access to information. The database management system (DBMS) provides the means for applications programs to use standard procedures to store and retrieve information. The RDBMS also provides operational support systems and database administrator (DBA) functions. This section addresses the operational roles of each aspect of the DBMS.

A COE database server provides data management services to all applications. In order to be useable it must be a stable, reliable operating environment that developers can design to and trust the consistency of its behavior. Database services include tools to support the management function and discretionary access to data based upon authorized permissions. This is governed by the following principles:

- Users will not need access to all applications.

- Applications will have multiple levels of database access that can be granted to users.

- When access to an application is granted to or revoked from a user, the corresponding database permissions are also granted or revoked.

COE database services are a federation of application-owned and common databases. Application segment developers control the data and structures that are specific to their segments and can change those data or its structure when necessary.

The common, corporate databases are owned by DISA; their data and structures are centrally controlled. These databases reside on the database server that provides various services to the applications, acting as database clients, within the network. The databases within a particular database server are isolated from each other, physically and logically, by being placed in separate storage areas and by being owned by different DBMS accounts. Database developers sustain this isolation by defining one or more database accounts to own their data objects and by allocating those objects to the owner accounts they have created.

This configuration, using the disk controller and drive analogy is shown in Figure 4-1. The core database configuration, containing the DBMS data dictionary and associated system information is part of the COE and is represented by the System Database. All other databases, whether provided by DISA, a developer, or Service/Agency, are included as “Component” Databases under the management of the database server. The set of component databases available within a particular database is determined by the applications set supported by that database.

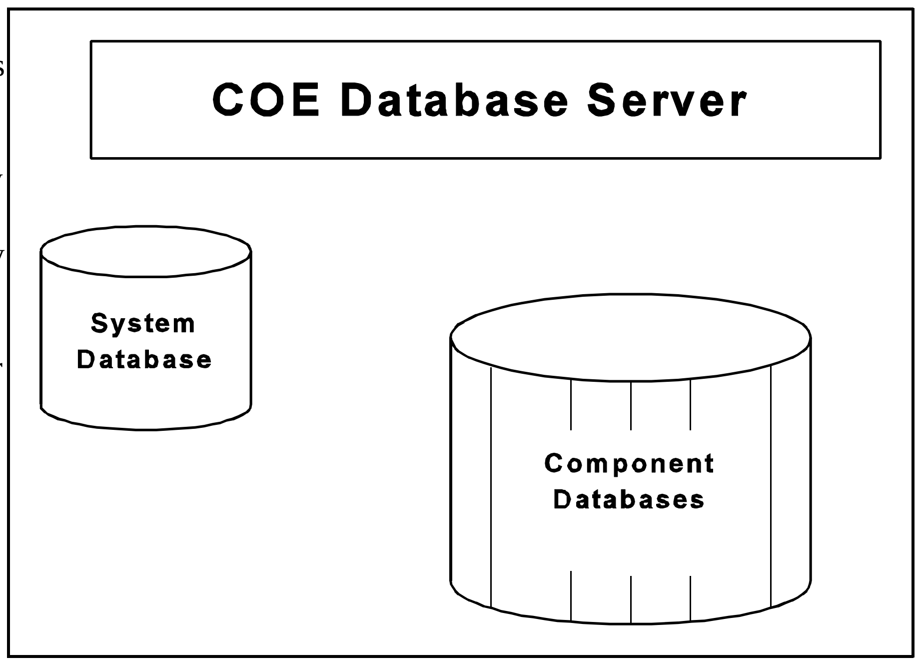


Figure 4-1. Database Server Model

All databases are shared assets, whether they are common or not. They are also dynamic because their data can change even while the structure remains static. Databases may be interdependent. Databases depend on the COTS DBMS service and are built within the particular DBMS constraints. Databases can be accessed by resident applications or by other applications written by database developers.

Users connect to the database server through the applications, possibly in multiple sessions. Each session must behave as if it is isolated from the rest of the system and knows of no data other than that belonging to the application it is executing.

4.2.2 Constraints on Database Developers

The developers of databases and applications accessing databases must conform to the conventions and constraints of the COE database server environment to prevent bypassing or corrupting its features. The combination of the database server configuration and the developers' implementations must ensure two things:

2. Each user connecting to a database through an application must function in the proper context for that application and database.

2. Each user's database connection must not interfere with any other user's connection to the same database or any other database.

The development and integration standards for COE databases support an evolving configuration of database services. In the past, each application had its own database and DBMS. With the implementation of DII COE, various disparate database servers were replaced by a single server running a single instance of the DBMS. Each application retained ownership of its database within the segment owned by the application instance, while sharing the DBMS service with the other applications' databases. Thus, the database server provides a shared database environment, allowing concurrent access to multiple databases with varying degrees of autonomy.

The single server, single instance data management service conserves system resources by not requiring multiple copies of the DBMS to execute simultaneously. It eases the system management burden by providing a single point-of-entry for all database management services. That single point of entry also simplifies application development. The benefits derived from the centralized database services do limit the freedom of developers by requiring database implementations that are compatible with the larger multi-database environment. This limits developers by constraining their databases to function within a consistent parameter of the administrative framework.

The primary consideration for developers is that their applications and databases no longer have exclusive use of the DBMS. Instead of being an application-specific data management tool, the DBMS is a central repository service that supports all DII applications' database requirements. As a result, developers can no longer customize or arbitrarily tune the DBMS to an application's particular behavior requirements. Any required "tuning" shall be designed within the applications while accessing the database using a common set of services (i.e., APIs). Any modifications to the DBMS will inevitably affect other applications and the databases. Similarly, the individual component databases are no longer the sole occupants of the DBMS. Developers must implement their applications and component databases so that they do not interfere with other programs sharing the same DBMS. When there are multiple segmented databases in a DBMS, applications can connect improperly to other databases. Developers must ensure that their applications connect only to the intended database. They must also design their databases access to maintain data integrity without reference to external applications.

In order for component databases to "plug and play" properly in a multiple-database server environment, they must conform to the programming conventions, disciplines and standards defined in this document and the DII COE *I&RTS* document. The sole objective is to support the independent development of maintainable databases that will function reliably within the larger multi-database system.

Developers must implement their databases such that the operational site administrators can manage the collection of databases. System and database administrators could not be expected to manage multiple databases, each with its own integrity rules and access methods. This means developers must adhere to common implementation methods.

4.2.3 Database Integration Requirements

The database server is the DII COE component that provides shared data management within all COE-based systems. Regardless of the COTS DBMS used to provide database services, the basic functions within the system remain the same. The support services provided are:

- Support independent, evolutionary implementation of databases and applications accessing databases.
- Manage concurrent access to multiple, independent and autonomous databases.
- Maintain integrity of data stored in the DBMS server.
- Provide discretionary access to multiple databases.
- Sustain client/server connections independent of the client application and database server hosts.
- Support distribution of databases across multiple hosts with replicated data and with distributed updates.
- Provide maintainability of users' access rights and permissions across the databases.

Database services within the COE are not restricted to a single vendor's DBMS. As a result, developers must implement their databases in such a manner that dependence on any particular DBMS vendor's product is strictly limited to assure multiple accesses to various databases and portability of the applications developed.

4.2.3.1 Evolutionary Implementation

The goal of an evolutionary implementation is to be able to incrementally develop, field, and improve software and information services. This "build a little, test a little, field a lot" philosophy applies to databases as well as applications. In the database context, the objective is to field the latest and best information structures and contents. Databases and applications should be able to evolve independently in principle. In practice they are not likely to because of the dependence of applications on the database's internal structure. Furthermore, the component databases are dependent on the DBMS for their implementation.

Database developers can still support evolutionary implementation by maintaining the modularity of their component databases. To achieve this goal, component databases must first coexist within the server without corrupting each other's data. This does not require databases to be isolated from each other. It only requires that all actions across database boundaries be intentional and documented. The COE architecture requires that segments not modify other segments. The same applies to component databases modifying or extending other databases. When a database does have a dependency on other database component, the dependency must be kept in a separate segment.

Component databases are dependent on the particular DBMS being used for the database server. The specific commands used for their implementation within the DBMS, and the environment it provides are both provided by the DBMS vendor. Database developers must be careful in their use of vendor-specific features so they do not create unintended dependencies on specific database management systems or, more importantly, particular versions of the DBMS while still taking advantage of the database server's capabilities. To accomplish this, developers shall separate DBMS-specific code from that which is transportable.

Most DBMS constraints also apply to applications accessing those databases. Application developers must ensure that applications connect through regular, documented APIs and shall not use vendor-specific DBMS features within their applications. This does not prohibit developers from using DBMS vendor-supplied tools that are approved and part of the DII COE or from accessing objects in other database segments.

4.2.3.2 Managing Multiple Databases

As stated earlier, the COE database architecture is a federation of databases with varying degrees of autonomy. Federated means that the component databases are collocated and share the same DBMS resources. They process data cooperatively but are not part of an overall schema. In some cases they may also share or exchange data. Autonomous means that each database remains an independent entity. Individual databases may be modified or upgraded without reference to others. They are also responsible for maintaining their own data access and update rules.

The federated architecture provides the same modularity within the database server that mission application segmentation does for the user interface. The set of databases available from any particular database server is tailored to the information needs of the individuals using that server.

Each component database must be implemented in a self-contained manner to assure interworking. This is not to say that a database supporting a set of applications should be self-sufficient. The modular database implementation goal is to limit the redundancy of information among component databases. Developers should not incorporate information in component databases that are already available from other, existing databases. Instead, being self contained means that each component database must contain all information needed to manage its objects and maintain their integrity.

4.2.3.3 Data Integrity

Data integrity addresses the protection of the information stored within a DBMS. There are three general circumstances that must be addressed:

2. The prevention of accidental entry of invalid data.
2. The security of the database from malicious use.
2. The protection of the database from hardware and software failures that may corrupt data.

The implementation of appropriate data integrity measures is the responsibility of the database developers using the features of the DBMS.

The database server is responsible for preserving the integrity of each component database and for preventing connections between an application and data that belong to any other application. COE-based systems are typically secure systems that contain and process classified data. The database management component must conform to the security policies and practices of the overall program. Otherwise, the database server supports the data access restrictions and integrity assumptions incorporated in each database.

The database server provides the basic functionality expected of a DBMS. It ensures the recoverability of failed transactions or of a crashed system. The "ACID" properties, for atomicity, concurrence, isolation, and durability, of database transactions are the responsibility of the applications accessing the server. However, supporting these transaction properties is the responsibility of the server. Developers must pay special attention to transaction isolation due to the multi-database configuration of most COE based systems.

Database developers are responsible for defining and implementing the integrity constraints of their databases. The database server is responsible for enforcing the developers integrity constraints when they are defined within the database. Application developers must ensure that their applications connect properly to their databases and do not connect improperly to anyone else's databases. Adoption of these practices protects all applications' data and allows the database server to maintain all databases reliably.

Within a component database implementation of data integrity takes the form of constraints and business rules. In the current context, constraints are defined as the rules within the database that govern what values may exist in an object. Business rules are those rules within the database that govern how data is updated and what actions are permitted to users.

Commercial database management systems were traditionally limited in their ability to support the variety of constraints and business rules needed in a database. This resulted in coding the required constraints and business rules into the applications, rather than in the database where they should be. The federated database architecture and the applications that maintain those databases are developed independently. This causes difficulty in ensuring uniform and consistent enforcement of those rules and constraints.

Henceforth, developers shall place their business rules and constraints in their databases rather than the applications. This will retain control of data maintenance access with the developers where it belongs and ensure that constraints cannot be bypassed. Developers have the knowledge of their constraints and business rules; DBAs and users do not.

Figure 4-2 shows an example of applications and database independence by placing constraints into the data access rather than in the applications. The placement of business rules and constraints in the database promotes client/server independence. The efficient implementation of constraints and business rules makes maximum use of current DBMS capabilities. When the rules are in the component database, the application is less dependent on the COTS product. This approach reduces network communications loading by allowing the DBMS, rather than the application, to enforce the rules. Checking rules within the database avoids passing multiple queries and their results over the network between the DBMS and the application.

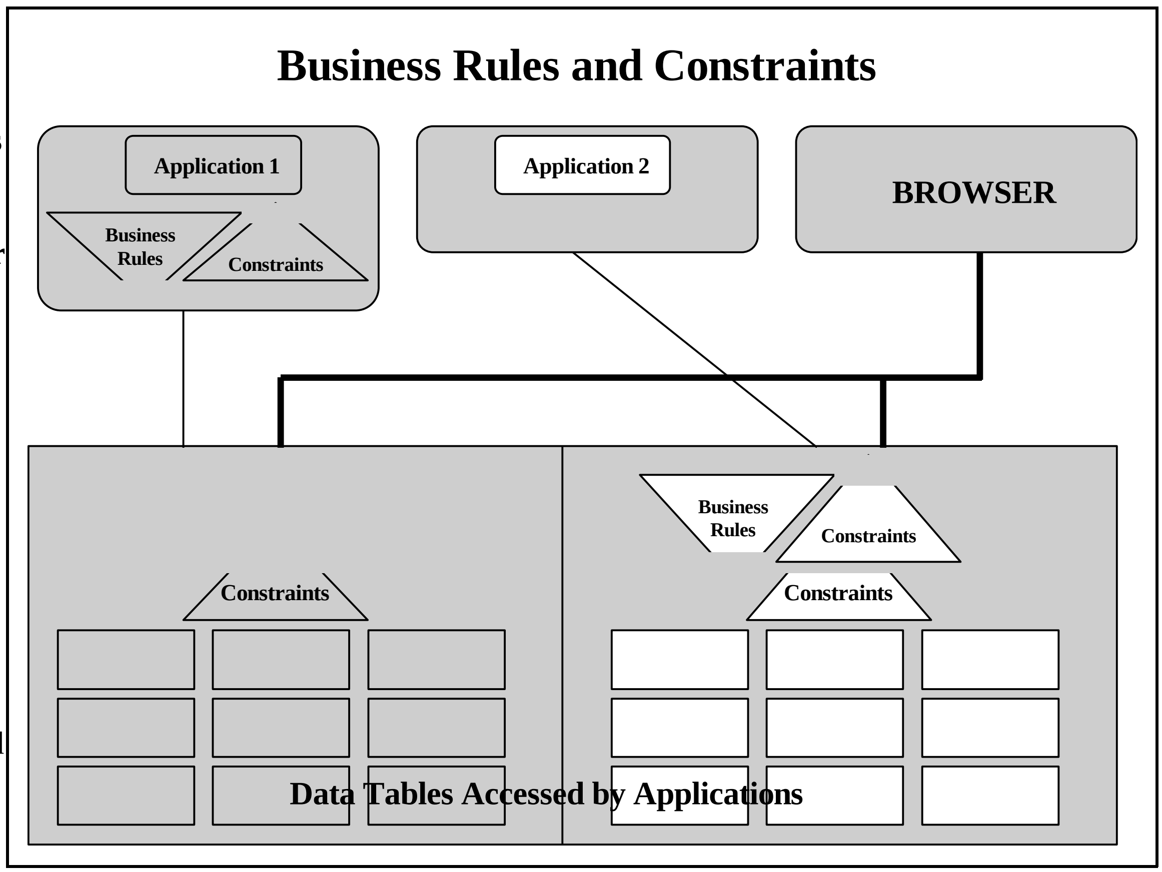


Figure 4-2. Business Rules and Constraints

4.2.3.4 Discretionary Access

Discretionary access addresses the selective connection of users to databases through the applications. Database access is discretionary because not all users have the same access privileges (or permissions) to use the applications. The objective of discretionary access is to ensure that users' database connections operate in the proper context for the applications. Users must be able to operate several different applications simultaneously. The DBMS server must manage each application's accesses to different sets of data objects. This provides the means to give an application entity or user selective access to specific tables with a particular access mode (read or write access). Since several databases may exist on the database server, each application must be written to access only the database it can access or it belongs to and each user application connection must have only the permissions needed for that application context.

There are three components to this issue:

- Session Management, refers to the ability of the DBMS to keep different connections separate.

- **Discretionary Access Control**, addresses the context of an individual connection.

- **Access Management,** deals with the requirements of system and database administrators to manage the accesses that are provided to users.

Data integrity and consistency can be compromised without the correct functioning of all three of these components. In order for this system to be useable, it must provide support to systems administrators as they manage users' discretionary access to subsets of applications and databases. See the latest version of the DII COE *I&RTS* for more detailed information.

4.2.3.5 Supporting Multi-Database Tools

Access to multiple databases is one of the major benefits that the COE brings to its users. Database browser tools, such as APPLIX, allow users to construct *ad-hoc* queries that span different subject areas and that are not supported by mission-specific applications. Such multi-database applications present special problems in the COE context. If the databases were read-only, browsers would not cause problems. However, many databases are designed to be maintained interactively using the applications associated with them. This means that users will have permission to write to databases while other applications seek access to the latest information requested by their queries. Those write permissions are potentially active when a user is using a database browser. That may mean the browser tool can also write to COE databases. Since the browser is independent of the applications designed for particular databases, it will be unaware of any constraints or business rules that are in those applications. Thus it could corrupt data due to its ignorance of the rules.

The key to ensuring database integrity in this case (as in all others) is the enforcement of constraints and business rules within the database. If the rules are part of the database, they cannot be bypassed by any applications accessing the database. While the database server may withhold write permissions from browser tools, maintaining the constraints in the database provides an extra measure of protection. This also supports the future employment of browser tools as multi-database read/write applications.

Another issue relates to understanding the context of a particular database. When users formulate queries that span multiple databases, they are likely to encounter differences in the way information is represented among those databases. This may lead the users to draw erroneous conclusions from their query results because they do not understand the different contexts between the databases. To limit such occurrences, component database developers shall provide comprehensive information on their databases to be incorporated in the DBMS data dictionary. This information is part of the database segment.

4.2.3.6 Client/Server Independence

The COE uses a client/server architecture that is also applicable to the database services. Developers must preserve the independence of their applications, functioning as DBMS clients, from the database server. Applications accessing databases must be built independent of the database server. The location of the DBMS application shall not be dependent on the database residency requirements to work correctly. Developers cannot assume that all operational sites will have a local database server. Further, where sites have a local database server it may be on a separate machine hosting that is dedicated to the DBMS, or the server may be collocated with the application on a single machine acting as the application server and the database server. To maintain independence and support the client/server architecture, applications cannot assume residency on the particular database server.

4.2.3.7 Distributed Databases

A distributed database is one whose data are spread across multiple sites. Data are replicated in a distributed database when copies of particular objects or records exist in more than one of those sites. Data are fragmented when they are split among sites. Databases are distributed (fragmented or not) to improve responsiveness and increase availability in systems that serve geographically dispersed communities. In some circumstances, databases are replicated to enhance their survivability. The implementation objective of any distributed database is to provide location transparency. This means that the user need not know where data reside to enable access to data or to process the data.

The COE provides distributed database management services for the developers of distributed databases so they can maintain location transparency and distributed transaction processing.

4.2.4 Guidelines for Creating Database Objects

This section provides guidelines for developers in creating their database segments. Its objective is to support consistency across different databases and improve the mutual independence of the database federation.

4.2.4.1 Database Accounts

Three categories of database accounts have been defined within the DII COE. They are:

- DBAs
- Owners
- Users.

Each category has different functions and levels of access to DBMS based functions.

The DBA accounts have access to all parts of the DBMS. They are to be used only for system administration. Their use by database segments is prohibited except during the installation process.

The owner accounts are the creators and owners of the data objects that make up an application server segment. The name must be unique within the COE community. Developers normally use the segment name or a variant of it as the owner account name. The segment prefix will also be used as the database schema name and will be incorporated in database file names as discussed below. Owners accounts must have their password changed after a database installation. Users shall not use the owner accounts to connect to the databases. Developers shall not grant the DBA privilege to owner accounts.

The user accounts belong to the individuals accessing databases. Each individual must have his/her own unique user account. Creation and maintenance of user accounts is a site DBA responsibility. Developers shall not assume the existence of particular users and shall not create user accounts. The creation of accounts that perform database services is an exception to the rule that developers not create user accounts.

4.2.4.2 Physical Storage

DBMSs provide file management transparency across multiple host compiler systems by hiding the details of file storage from the database's data objects. At the same time, however, the placement of data objects on physical storage devices has an impact on system and database performance due to disk contention and other file system access issues. There are two standards for storage: 1) Data Store/File Standards and 2) Data Storage Standards.

The DBMS managed components of a database segment can be grouped into functional sets based on their use within the segment. These functional sets are defined as a data store. Data stores are physically kept in database files whose implementation varies depending on the DBMS being used. A segment's database will normally consist of two functional sets (data and indexes) and hence two data stores. The data store identifier will incorporate the database segment prefix and the function of the data store.

Developers shall define one or more data stores for their database segments. The objective is to allow data files to be spread across multiple, physical storage devices based on the data store's function within the DBMS.

Data store names must also be associated with the segment and function. Most applications will have either two or three data stores: data, indexes, and (if needed) static data.

For more information on this subject see the *DII COE I&RTS* and the *DII COE I&RTS Database Development and Integration Standards*.

4.2.4.3 Database Objects

The definition of a database schema - the set of data objects, their interrelationships, constraints, and rules for access or update - is the responsibility of the developers. Developers shall not duplicate data objects that are part of the corporate databases provided by DISA. Where possible and appropriate, developers shall take advantage of and share objects belonging to other databases within DII. Developers shall provide definitions for their schema components for inclusion in the DBMS data dictionary. In addition, narrative information on all these databases should be provided during Segment Registration so developers can access their definitions in the COE On-line Services.

Other considerations related to database objects are covered in more detail in the *DII COE I&RTS*. Some of the additional requirements relate to the following:

Database Tables:	Objects that store data records.
Data Elements:	Columns or fields within a schema that are grouped together into tables.
Data Views:	Stored queries that do not actually contain or store data, but derive their data from the tables on which it is based.
Rules for DB Objects:	Maintain database integrity through the enforcement of the constraints and business rules of the database.
Constraints:	Restrictions on data elements with respect to the values they may contain.
Stored Procedures:	Consists of a set of DBMS commands that are stored in the database and can be invoked by an application to perform a task, or a set of related tasks.
Triggers:	Procedures that are automatically executed when a triggering event occurs on the associated table.
Indexes:	Optional structures associated with a table, that is used to quickly locate rows of that table, or to ensure that a table does not contain duplicate values in specific columns when a uniqueness constraint cannot be used.

4.2.4.4 Database Roles

A database role, in the general sense, is a group of access privileges on database objects. These roles implement the discretionary access controls. Database roles also simplify the management of user privileges within the DBMS. They are created by the database segment developer or the developers of application accessing databases to define sets of access privileges that can be given to users by their sites' DBA's. Role names will be meaningful (the database or application name should be part of the group or role name) without using reserved words. Developers should strive to associate roles and their privileges with the applications accessing the database. Each role should have only the privileges needed by the application it supports.

When applications are not developed by the database segment developer, the application developers are responsible for creating the roles required to access the database through their applications. The access requirements for such roles must be defined by the application developers and included in the information provided during Segment Registration. The permissions required by application's database roles are subject to review by DISA and by the associated database segment's sponsor. These roles will be granted the privileges required to run the application. These privileges may include: delete, insert, select, and update for tables and views; and execute for procedures, functions, and packages. Grants of privileges to roles are discussed in the next section.

In order for the privileges on objects to be assigned to a role, the grantor must have permission to do so, and those database objects must exist. When application developers define database roles to support their applications and those roles are not part of the principal database segment, the roles and the grants that enable them become part of a database segment that is dependent on the database segment or segments that create the referenced objects.

Database roles will not be granted to DBAs. Their administrative privileges already allow them to grant roles to users without owning the roles. The database roles that are part of the COTS DBMS will not be altered by developers.

4.2.4.5 Grants

Grants are the permissions on database objects that allow users to access data they do not own. When a database object is first created, the only account that can access its contents is the owner of that object. Users must be explicitly granted permission to access an object. Privileges that can be granted include: delete, insert, select, and update for tables and views; and execute for procedures, functions, and packages. Privileges that should not be granted include index and alter for tables. Grants allow the DBA to administer and the DBMS to enforce the discretionary access controls required. As discussed in the section on database roles, developers should grant only the minimum set of permissions needed for the applications that access their databases. Grants should be made to roles/groups and not to individual users.

4.2.4.6 Inter-Database Dependencies

Interdatabase dependencies occur whenever database objects in a segment are dependent upon objects in some other database segment. A database object is a dependent object if it references any other object(s) as part of its definition. When a dependent object is created, all of its references to other objects must be resolved. If it has dependencies on non-existent objects, the dependent object may not be created or it may have to be validated when the objects it references come into existence. The creation of a dependent object may also fail if its owner does not have the appropriate access to all referenced objects. If the definition of any of the referenced objects is altered, the dependent object may not function properly or may become invalid. Implementation of inter-segment dependencies may occur through the use of dependent objects, constraints, and database roles.

For more detailed information on Database Concepts, see the *DII COE I&RTS* and the *DII COE I&RTS Database Development and Integration Standards document.*

4.3 Common Desktop Environment

The Common Desktop Environment (CDE) is a graphical user environment for UNIX workstations. This section **is not applicable to Microsoft New Technology (NT)** which uses the ‘Windows” a similar windowing system interface. CDE helps in bringing every application, network resources, and Internet services into one integrated Graphical User Interface (GUI) throughout the systems. It unites the UNIX desktops under a single graphical environment, providing consistency across platforms and making it an ideal homogeneous solution across different workstations.

4.3.1 Introduction to CDE

The Common Desktop Environment (CDE) is was jointly developed by IBM Corporation, Hewlett-Packard Company, Novell Inc., and SunSoft Inc. They defined a common set of application programming interfaces (API) for a CDE that can be implemented in an operating environments that is able to support X-Window desktops and OSF /Motif.

The Common Desktop Environment gives end users an easy graphical user interface (GUI) across workstations irrespective of operating systems and hardware. CDE provides end users with a consistent GUI across X- terminals and PCs, with a single set of APIs for IBM, HP-UX, AIX, Solaris, UnixWare, and all UNIX system based platforms. CDE provides end users a transparent access to data and applications from anywhere in the network.

4.3.2 CDE Tools

CDE is available through TriTeal Enterprise Desktop product which includes the following applications development tools providing features described below:

Mail Tool - compose, view, and manage electronic mail through a GUI. Allows the inclusion of attachments and communications with other clients through the messaging system.

Calendar Manager - manage, schedule, and view appointments, create calendar, and interact with the MailTool.

Workspace Manager - organize work into multiple workspaces and navigates between the workspaces using the front panel.

Editor - create, update and manipulate text with common functionality including clipboard interaction with other applications via Motif Clipboard and ICCCM Primary and Secondary selection mechanisms.

Terminal Emulator - emulate X-term-like terminals that supports terminal emulation of ANSI X3.64-1979 and ISO 6429:1992(E) compliant terminals.

Calculator - provide standard calculator functions, using screens and keyboards for inputs and print.

File Manager - interface with file system and graphical represent all data objects, provide "drag-and-drop" functionality between objects and between cooperating client applications, and a general file type association database.

Print Manager - (simple GUI print job manager) schedule and manage print jobs on any available printer.

Help System - provide context sensitive graphical help system based on Standardized General Markup Language (SGML).

Style Manager - set preferences interactively for a session, such as colors, backdrops, and fonts using a GUI interface.

ToolTalk - a messaging service that allows communication between applications.

Session Manager - save the current desktop configuration upon logout and restarts the entire session exactly as it was, including the application that were running and the location and sizes of the windows.

Login Manager - control access with system password protection and support password aging and Kerberos.

Application Builder - provide tools for constructing applications to run under CDE (TED).

Under the DII concept (*User Interface Specifications for The DII*, Version 2.0), the standardization of CDE can increase the user productivity, reduce training requirements, and increase the efficiency in the development of individual applications. The commonality in "look and feel" is a key element of usability, the concept of an application is central to the user understanding a system's capabilities and how to interact with them.

An application can be viewed as the software available to the user to perform a set of related tasks. This software is visible to the user as a collection of window families, each providing the functionality (in terms of objects and information) needed to perform a particular task. In addition, it is possible for applications to share the services provided by a segment when the applications perform common tasks.

There is a set of specifications provided in the style guide which emphasize a consistent user interface as described in the Mayhew's book *Principles and Guidelines in Software User Interface Design* (p.97), that provides:

- Consistent location of certain types of information on screens,
- Consistent syntax of commands in a command language,
- Similar execution of analogous operations in different applications,
- Consistent design of command names and abbreviations,
- Consistent grammatical form of error messages and instructions,
- Consistent design of captions and fields on forms and displays,

- Consistent dialog style for different functions, and
- Terminology consistent with the users' existing vocabulary.

4.3.3 DII Style Compliance Requirements

Compliance with the specifications in the *DII COE Style Guide* is required in the development of CDE. All software is expected to comply with the style guide specifications, with deviations occurring only when called for by operational requirements and approved by DISA.

The DII style guide provides guidelines that apply to new development of the GUI in addition to the existing DoD *Human Computer Interface Style Guide*. The Style Guide covers the following topics:

- Domain level specifications for the 'look and feel' of the software:

  - Buttons
  - Scrolled windows
  - Application Icons
  - Text fields
  - Command box, selection box and message box
  - Frames, labels and panel window
  - Font sets
  - Color sets

- Compliance levels - Level 8 is 'Full COE Compliance.'

- Guidance related to user interface internationalization, design of on-line user documentation, and user interface functionality in common support applications.

Appendix D and H of the *DII COE Style Guide* map specifications to each of the style-related items included in the COE compliance checklist published in the *I&RTS* document.

4.4 COE Security

Security of the operating system and COE application including the development environment is necessary to maintain the integrity of applications and data. The physical environment in which the system resides is mission dependent and the responsibility of the individual facility. The following outlines the security measures incorporated into the DII COE and includes on-line system services as well as security considerations for application segments.

4.4.1 Overall Security Features

COE on-line services are separated onto a classified and an unclassified system. The systems, whether classified or unclassified, use a secure operating system, database, and network software. Auditing is enabled to record system access and to other security relevant operations. Additional security features are implemented to:

- ensure software integrity,
- prevent interception or eavesdropping on data transmissions, and
- ensure separation of classified versus unclassified information, segments, and data.

The classified and unclassified components reside on physically distinct computer systems separated by an air gap. The unclassified system is available via Internet and is generally available to any interested party. The classified system is accessible only via SIPRNet, and only by authorized users.

Unauthorized access to the system is prevented through a layered approach. Firewalls are implemented as the first layer of protection. Secure routers provide IP address filtering and port access to limit access only to authorized workstations. Features are also implemented to restrict services that can be requested or granted to further protect the system from unauthorized access.

User authentication is based on a combination of a manual registration process, an authorized IP address, and password protection. Passwords are required to initially log onto the system, but are further required to log into the software repository and to access Netscape browser services.

Public key encryption is used to protect segments in the software repository. Encryption and compression are both used to protect data during transmission over the network to prevent unauthorized modifications.

Certain information, such as system problem reports or project status, is not necessarily classified. However, such information is still sensitive and needs to be controlled. Public and private views are implemented to provide this measure of protection.

4.4.2 Account Groups

Users are normally assigned individual login accounts with configuration files that establish a runtime environment context. These configuration files must be set up and established for each user of the system. An account group segment is a template used within the COE for setting up individual login accounts. Account groups contain template files for defining what COE processes to launch at login time, what functions are to be made available to operators, and preferences such as color selection for window borders. Account groups are described further in Chapter 5 of the DII COE I&RTS document.

Account groups can be used to perform a first level division of operators according to how they will use the system. This technique is used in the COE to identify at least five distinct account groups:

- Privileged Operator (e.g., root),
- Security Administrator,
- System Administrator,
- Database Administrator, and
- Non-Privileged Operator.

Other account groups may exist for specialized system requirements, such as providing a character based interface, but all account groups follow the same rules. Within an account group, subsets of the available functionality can be created. These subsets are called *profiles*. An operator may participate in multiple account groups with multiple profiles, and can switch from one profile to another without the need to log-out and log-in again.

4.4.2.1 Privileged User Accounts

Most operating systems provide a ‘super user’ account called root. Its use is normally restricted to knowledgeable systems administrators because serious damage can be done to the system if used improperly. Security requirements also dictate careful control and auditing of actions performed when operating as a privileged user.

The COE design philosophy is to not require the use of a privileged user account for normal operator activities. Certain processes (such as setting the system time) can not be performed without superuser privileges, but such privileges are given to the process, not the user, and only for the period of time necessary to perform the required action. Root level access is **not** provided to the user for such actions.

Normal operation does not require a command-line level access to root. For security reasons, command-line access is expressly prohibited unless prior approval is granted by the DISA Chief Engineer. However, a root account is preserved in the system for use by trusted processes, for unusual system administration tasks or installations, and for abnormal situations where ‘all else fails.’

4.4.2.2 Security Administrator Accounts

Security in the COE is implemented through a Security Server and through a special trusted security client. This client is the Security Administrator application, which is an interface to the Security Server, that allows a Security Administrator to monitor and manage security. Precise functionality of the Security Service is hardware-dependent because different approaches have been taken by different vendors to provide security.

The Security Service is loaded as part of the bootstrap COE. The Security Administrator application is designed to be made available to only a restricted group of operators. Available functions include the following:

- Ability to create individual login accounts
- Ability to create defined operator profiles
- Ability to customize menus by operator profile
- Ability to export accounts and profiles to other LAN workstations
- Ability to review and archive the audit log.

The Security Administrator software provides the ability to describe objects (files, data fields, executables, etc.) which are to be protected from general access. This information is used to create profiles which limit an operator’s ability to read or modify files. Applications may query the security software to determine the access permissions granted to the current user. The `Permissions` file is the mechanism for segments to describe objects and permissions to grant or deny for the objects.

This descriptor is a sequence of lines of the form:

```
object name:permission abbreviation:permission
```

object name is the item to be controlled, *permission* is the type of access to grant or deny (add, delete, read, etc.), and *permission abbreviation* is a single character abbreviation for the permission.

Permission abbreviations specified for an account group must agree with all segments which become part of the group. The following are reserved abbreviations and their meaning:

A - Add
D - Delete
E - Edit
P - Print
R - Read
V - View
X - Transmit

Segments may use additional abbreviations as required.

For example, suppose a segment generates reports that are to be protected. Permissions to grant or deny for reports are delete, print, read, or archive. The proper `Permissions` file is (Z is used to indicate archive permission in this example):

```
Reports:D:Delete:P:Print:R:Read:Z:Archive
```

If the `Permissions` file is missing, the security software will report that for the requested object, no access permissions are to be granted.

4.4.2.3 System Administrator Accounts

The System Administrator Account Group is a specialized collection of functions that allow an operator to perform routine maintenance operations. This software is designed to be made available to a restricted group of operators. It is loaded as part of the bootstrap COE because it contains the software required to load segments. Functionality provided includes the ability to:

- format floppy disks
- install and to remove segments
- set workstation name and IP address
- install and configure printers
- create and to restore backup tapes

- shutdown and to reboot system
- configure DDN host tables
- configure and manage the network.

4.4.2.4 Database Administrator Accounts

The Database Administrator Account Group is to be used by those individuals responsible for performing routine database maintenance activities such as backups, archives, and reloads. The specific capabilities are dependent upon which commercial relational database software is in use and upon tools provided with these commercial products.

Functions included within this account group are the ability to:

- archive and restore database tables
- import and export database entries
- checkpoint and journal database transactions.

4.4.2.5 Operator Accounts

Most operators will not require, nor will site administrators grant access to, capabilities described in the previous subsections. Most system users will be performing mission specific tasks such as creating and disseminating Air Tasking Orders (ATOs), preparing briefing slides, performing *ad hoc* queries of the database, participating in collaborative planning, etc. The precise features available depend upon which mission application segments have been loaded, and the profile assigned to the operator.

4.4.3 Security Considerations for Segments

COE-based systems typically operate in a classified environment. Therefore, security considerations must be addressed both by the COE and the segment developer. This section describes the security implications from a runtime environment perspective. It does not address procedural issues such as proper labeling of electronic media, requirements for maintaining paper trails showing originating authority, etc.

This section is evolving as security policies are developed for GCCS and GCSS, and as legacy systems are migrated to the COE. Further guidance will be issued as appropriate. Refer to the DISA Chief Engineer for specific security concerns, or for guidance in segment development in addition to the information contained here.

4.4.3.1 Segment Packaging

Segments shall not mix classification levels within the same segment. It is permissible to create an aggregate that contains segments that are at different classification levels, but the parent segment must dominate the security level of any child segments.

Features that are not releasable to foreign nationals shall be clearly identified through documents submitted to DISA when the segment is delivered. Software and data which contain non-releasable features shall be constructed so that the features may be removed as separate segments.

All classified data shall be constructed as separate segments. Developers shall submit unclassified sample data to DISA, as a separate segment, for DISA to use during the testing process.

4.4.3.2 Classification Identification

All segments shall identify the segment's highest classification level in the Security descriptor. Developers shall submit documentation to DISA which clearly identifies what features are classified, and at what classification level.

4.4.3.3 Auditing

Segments which write audit information to the security audit log shall include the segment prefix in the output. This is required so that audit information can be traced to a specific segment.

4.4.3.4 Discretionary Access Controls

Developers shall construct their segments so that individual menu items and icons can be profiled through use of COE profiling software. The profiling software allows a site administrator to limit an individual operator's access to segment functions by menu item, or by icon.

4.4.3.5 Command Line Access

Segments shall not provide an xterm window or other access to a command line, unless prior permission is granted by the DISA Chief Engineer. Segment features should be designed and implemented in such a way that operators are not required to directly enter operating system commands. When command line access is granted, it shall not be "root" access (e.g., privileged user). Situations requiring superuser access shall require the operator to log in as root.

Segments which provide command line access shall audit entry to and exit from the command line access mode. Entry to command line access mode shall require execution of the system login process so that the user is required to enter a password.

4.4.3.6 Privileged Processes

Segments shall minimize use of privileged processes (e.g., processes owned by root or executed with an effective root user-id). In all cases, privileged processes shall terminate as soon as the task is completed.

4.4.3.7 Installation Considerations

Segments shall not require PostInstall, PreInstall, or DEINSTALL to run with root privileges unless permission to do so is granted by the DISA Chief Engineer.

Segments shall not alter the unmask setting established by the COE.

4.4.3.8 File Permissions

Segments shall not contain any files directly underneath the segment's assigned directory. The reason for this restriction is that such files can be modified by an unauthorized user if the directory permissions are set to octal 777 (as the COE requires them to be for certain directories).

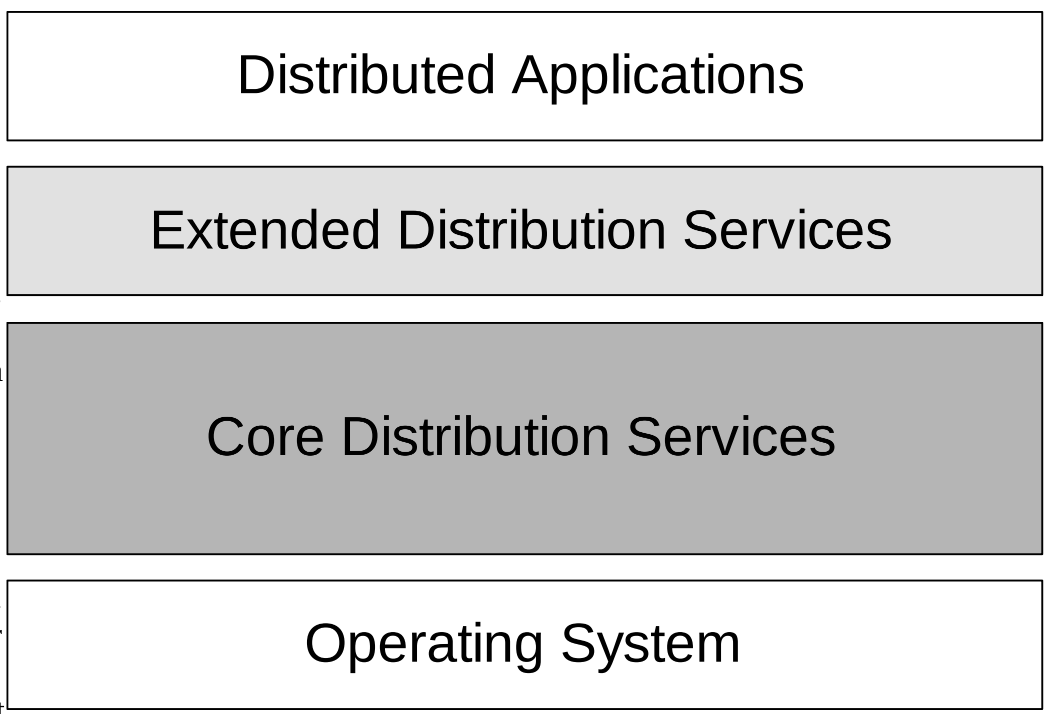
Segments shall not place any temporary files in the directory pointed to by TMPDIR unless deletion, alteration, or examination of such files by another segment or user poses no security concerns.

4.4.3.9 Data Directories

Segments which contain data that must have world access privileges along with data that must **not** have world access privileges shall split the data into separate directories underneath the segment's data directory. File permissions on the separate directories must be set to prevent unauthorized access to data files.

4.5 COE - Distributed Computing Environment

Distributed Computing Environment is Open Systems Solution supported on virtually every computing platform. It is available from desktop to mainframe systems. Its unique "Single Source" code is the basis of its multi-vendor support resulting in comprehensive interoperability and supports a verifiable branding program.



DCE is available from approximately 50 vendors who cooperatively developed the specifications in Open Software Foundation. The specifications are freely available to users and suppliers of DCE. The DCE technology was acquired through a open selection process and its future direction is set through open technical forums of the Open Software Foundation (OSF).

4.5.1 DCE Services

The DCE was adopted for use by DoD in the DII COE. It is based on DCE Threads which provide an advanced programming paradigm for distributed computing. DCE includes a “Light-Weight” process based on Remote Procedure Calls (RPCs). It easily adapts to a multiprocessing operating system while taking advantage of multi-processing technology. It supports parallelism on certain operations. It is highly portable across multiple platforms due to its common technology source. DCE is based on IEEE POSIX standards (PASC P1003.x). While DCE is entirely written in “C”, a very small portion is written in assembler. It runs on non-threaded kernels as well as threaded ones. DCE is built to support exception handling in Distributed Processing Applications.

The DCE Services are represented in Figure 4-3 below.

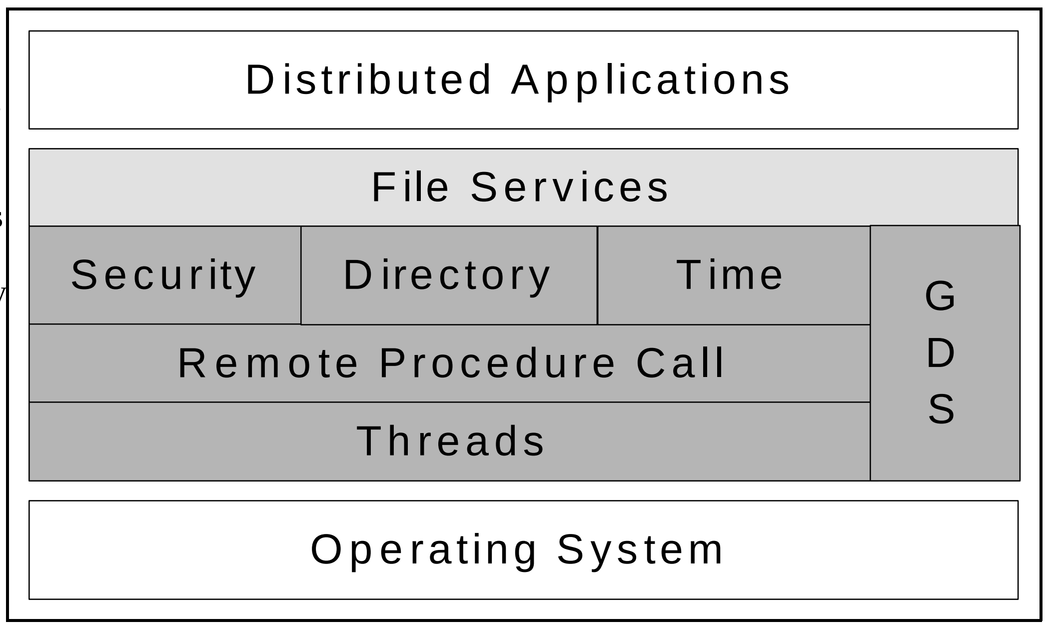


Figure 4-3. DCE Services

DCE services consist of Threads, RPC, Security, Directory, Time Services, File Services and Global Directory Services based on X.500 distributed directory standards. These services reside on the platform operating systems and use the available communication services to tie to other systems.

4.5.2 What is an RPC?

COE-DCE’s Remote Procedure Calls (RPC) is a traditional evolution of Inter-Process Communication (IPC) facilities used in large systems using multiple background tasks. The IPC was used for inter-task communications. In the evolution of networked systems tasks residing over a telecommunications line were remote procedures executed by various calls. Hence the

RPC is a facility for the activation of remote processes in a remote system over communications services.

RPC is an extension of a well understood programming models. It provides for network transparency and is transport independent. That is, it can run on TCP/IP, OSI, or other commonly standardized protocols. The RPC is data presentation independent due to its abstracts syntax notation. The networking code (stubs) are generated automatically, and the interfaces are specified in Interface Definition Language (IDL). RPC is based on mature technology that is reliable regardless of the communications transport being used. It allows for concurrent RPCs to run on a client and its' communicating servers. It integrates well with various naming and security conventions and requirements. DCE based RPC is a framework for future distributed Object Oriented programming.

4.5.3 The Cell Directory Services

DCE uses Cell Directory Service (CDS) to provide location transparency of services to the application developer. Servers can move physically, but their name always remains the same. The CDS facilitates scaleability by way of:

- partitioning,
- replication,
- client-side caching.

CDS provides very efficient intracell lookups and supports intercell directory lookups through DNS and X.500 directory services. CDS has the ability to support global namespace and transparent access to all domain resources across the globe without loss of local autonomy.

4.5.4 Distributed Security Services

The DCE Security Services (DSS) provide for a 'Single', network-wide registry with the ability to support single sign-on extensible to security, database access and credentials verification. DSS is scalability through use of replication, client caching, and inter-cell operations. It provides robust services by way of Authentication through Privileged Attribute Certificates (PAC) and POSIX Access Control Lists (ACL). This is done both using intracell and intercell security mechanisms. DSS also provides for message encryption services using Data Integrity and Privacy. It supports powerful delegation support capabilities.

4.5.4.1 DCE - DSS Registry

DSS provides for a single registry within a network domain consisting of:

- Principals
- Groups
- Organizations
- Accounts
- Policies and Properties, and an

- Extensible Registry Schema.

The DCE Privilege Service protects client credentials with signature inside of the security ticket. It prevents forgery of credentials by would-be attackers. The credentials are implemented using Privilege Attribute Certificates (PAC) defined into Cells, Users, and Groups. The Credentials and PAC are based on IEEE P1003.6 (Security Expert Group) ACL Standards. Its extensible attributes provide tracing of all delegations and manage the intercell operations. This provides a comprehensive security for distributed applications development in a DII COE environment using the DCE Access Control Model.

DCE provides a superset of POSIX ACL entry types. Users may obtain their credentials in the form of a PAC. The PACs are compared against ACLs to determine authorization.

4.5.5 DCE Distributed Time Service

DCE's Distributed Time Services provide a service essential for application synchronization in a controlled transaction environment. It provides scalability through a Hierarchical propagation scheme, a Fault tolerant fallback configuration, and can Import time from a variety of sources such as NTP, WWV, Spectracom, and GPS.

The DCE security services uses DCE authentication and access controls using Time as an interval, not a constant. It provides a new set of APIs and call library integrated with all DCE core services. Users of DCE DTS can use the Universal Coordinated Time (UTC).

4.5.6 Definition of a DCE Cell

A DCE Cell is a collection of users, resources and/or services. Typically within a cell users make use of resources. The resources are provided by the servers. One or more servers may implement a Service. A cell may be a collection of machines running DCE services. There may be three to 15 Security, Directory and Time Servers within a cell and approximately 150 clients per server. This is equivalent to having 0 - 25,000 entities per cell. Machines within a cell represent an administrative domain.

4.5.7 Distributed File Services

DCE provides for Distributed File Services (DFS) which makes a heterogeneous network look and act like a single system. DFS is scalable through the use of Local Caching. A Controlled replication mechanism and 'cloning' is provided which allows for global name space and a Global file naming and access mechanism. It provides full UNIX read/write consistency semantics and file access security is integrated with DCE security using the ACL support.

4.5.8 DCE Role in DII COE

DII COE version 2.0 includes all DCE Core components as follows:

- Directory Services

- Security Services
- Threads
- Time Services
- RPC
- COE does not include DFS (only available for some user communities).

DCE provides a secure distributed computing infrastructure for DII developers who want to begin building on COE. The primary benefit of DCE is the increased security capability.

4.5.9 Client Binding API (COE 2.0)

The DCE Client Binding API is delivered in COE version 2.0. It locates a server following *I&RTS* guidelines and provides authenticated access. The follow on guidance will establish a knowledge base by gaining and sharing knowledge and experience. COE will establish DCE administrative infrastructure by providing training to administrators with related operational procedures, security and establish guidelines for the Distributed File System.

4.6 COE Communications Services

This section covers the COE Communications Services which is a component of the DII COE. It defines and covers the various services and related topics important for COE developers and programmers requiring information exchange services between specific systems or for developing distributed mission applications.

4.6.1 Communications Services Overview

DII COE Communications Services provide the developers of COE components various capabilities for the exchange of information across distances or within a given space. It consist of Multi-Level Security (MLS) and provides for guidelines for secure communications capabilities. Additionally it defines the boundaries between applications and communications as well as the responsibility of the programmers with respect to moving data between systems.

Where there is a concentration of communications services, the communications servers are used to distribute the bandwidth required between backbone systems and local area communications services. The communications servers are categorized into two distinct servers: the Network Servers and the Channel Specific Servers.

COE developers requiring DII COE Communications Services have multiple choices of data transport services. These are categorized into Message Transfer Services, Network Services and Information Transfer services. The relationships of communication with respect to Other COE Areas are further defined below.

4.6.2 Communications Services in the DII COE Model

The Communications Services are both part of the Support Applications, Platform Services, and the DII COE Infrastructure Services. Additionally, the communications services reach to the external environment using the physical media facilities of the communications controllers, the wires and the various cables. The following figure depicts the position of communications services within DII COE Model.

Figure 4-3 shows in shaded area the communications services and their relative position within the DII COE model. The COE Communications Services include filing, messaging, teleconferencing, distributed computing and multi-media services. The physical media shown in this picture relate to SIPERNet and NIPERNet.

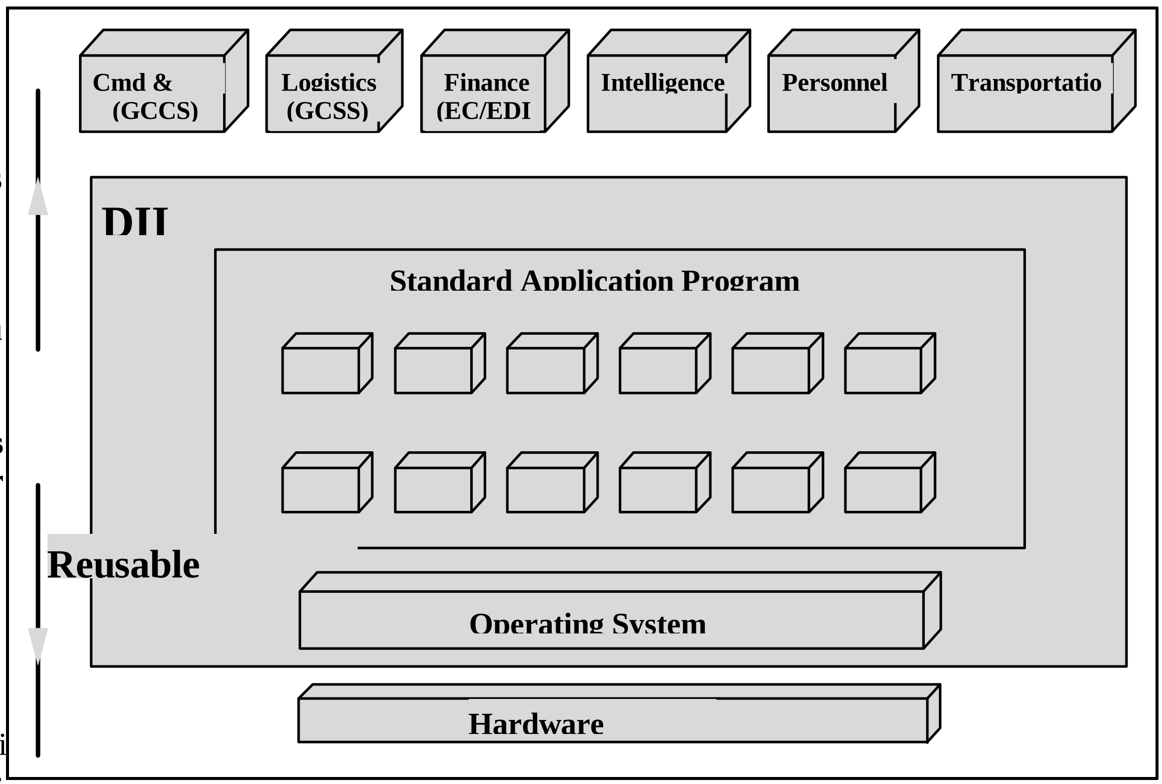


Figure 4-3. DII COE Communications Services Position

4.6.3 Types of Communications Services

There are various types of communications services available to the developers and continuous progress is being made into advanced telecommunication services such as multi-media (MM) and videoteleconferencing (VTC) for desktop (PCs) COTS products which will become part of DII COE in the near future. Currently there available services in messaging, file transfer, data exchange and distributed computing. The Defense Messaging System (DMS) is a secure messaging communications service in process of being implemented and will be used as a common electronic mail facility with various media attachments.

4.6.4 Communications Servers

Most 'Information Transfer' capability is supported in the DISA network model through a network server or 'Channel-Specific' servers. All tactical and legacy communications are supported through Channel-Specific servers for voice communications using the Plain Old Telephone Services (POTS) and the tactical 'Combat Net Radio.' It is expected that the latter will also provide visual communications which include video teleconferencing (VTC) and 'Shared Screen' capabilities.

DII COE Communications Services provide various modes of communications between parts of distributed applications or for information transfer services. These are high level communications protocols, commonly referred to as TCP/IP, such as Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and terminal access via Telnet. These communications services provide data transmission using secure, reliable, transparent, end-to-end data communications.

4.6.5 DII COE Infrastructure Services

The Infrastructure Services for communications are a set of coordinated services over various media such as twisted pair, copper and fiber cables as well as truck lines supporting connectivity and data exchange between one DII or GCCS system or workstation from one site to another.

The controlled mechanism for handling systems extensions and customization of services is made to fit the applications requirements. The COE programmer does not need to know about the details of the communications services it requests. The COE kernel and related components will provide coordinated and certifiable communications services where appropriate.

4.6.6 Communications Services Security

The availability of security services in the use of DII COE communications services include:

- Data Confidentiality
- Data Integrity
- Peer Entity Authentication
- Data Origin Authentication
- Access Control
- Non-Repudiation Security.

These security services are provided using the DII COE Communications Services tools and APIs as appropriate.

4.6.7 Communications Services Boundaries

The boundary between application components and the communications services are defined for the COE programmer to assure a sound level of understanding for programming requirements in the transfer of data between systems. These are dependent on the type of applications being built and the services required from the communications protocols. The boundary may be defined in several ways. In some instances the "Wire" on one end may be the defined service boundary between the applications and the communications services while in others the memory buffer or the consumer of service, such as the client application or service, on the other end define the boundary. In short the "boundary" will define the "Responsible Party" for getting data off the wire or sending the data across the wire.

Another aspect of communication service is the responsible party for "Bounding the Data Into Messages" where the "Message Unit" is defined by the context of the communications channel. The appropriate interpretation of the data is primarily the responsibility of an application in the client system or an application servers. This includes character conversion, data parsing and context management.

4.6.7 Communications Servers

Communications Servers provide Communications Services to a cluster of communications users. The servers provide the capability for clients and servers to communicate with each other and with external systems. A typical example of a communications server is the Message Transfer Agent (MTA) within a DMS or X.400 messaging system. In such an instance, the server is relaying messages received from various users located on various LANs to the next hub or node outside of its domain of control. In most instances such servers are also a gateway with secure access or "firewalls" to protect its assets. A server is named based on its transport medium: typically a network server facilitates LANs to LAN and LAN to WANs interconnection while a channel-specific servers will use parallel or serial media.

In most instances the mission applications or clients application is not concerned with the transport medium being used by the communications services. Clients and servers do not need to know the location of the destination server. This is done by the network or communications services. However, servers must know how to communicate with each other (or other servers).

4.6.8 COE - Network Servers

In each of the DII, GCSS or GCCS network configuration, such as a LAN, there will always be at least one network communication server. Currently the typical services provided by the network servers to DII COE developers is the higher level services of electronic messaging, file transfer, remote terminal access and Web Information Access. The current protocols associated with these servers are: SMTP, FTP, Telnet, and HTTP.

Multiple network servers may be used to support high bandwidth requirements for multiple systems using lower bandwidth links. In such configurations a channel-specific servers (CSS) also known as a sub-server may be used to multiplex lower bandwidth channels into higher bandwidth truck lines. The CSS is used to transfer data to external devices or systems which are **not** connected to a LAN (i.e., radio, cellular, mobile, etc.).

4.6.9 DII COE Notional Communications Architecture

The DII COE communications architecture is based on a notional network server model. This means that various applications services reside on servers and provide various services to clients and other servers.

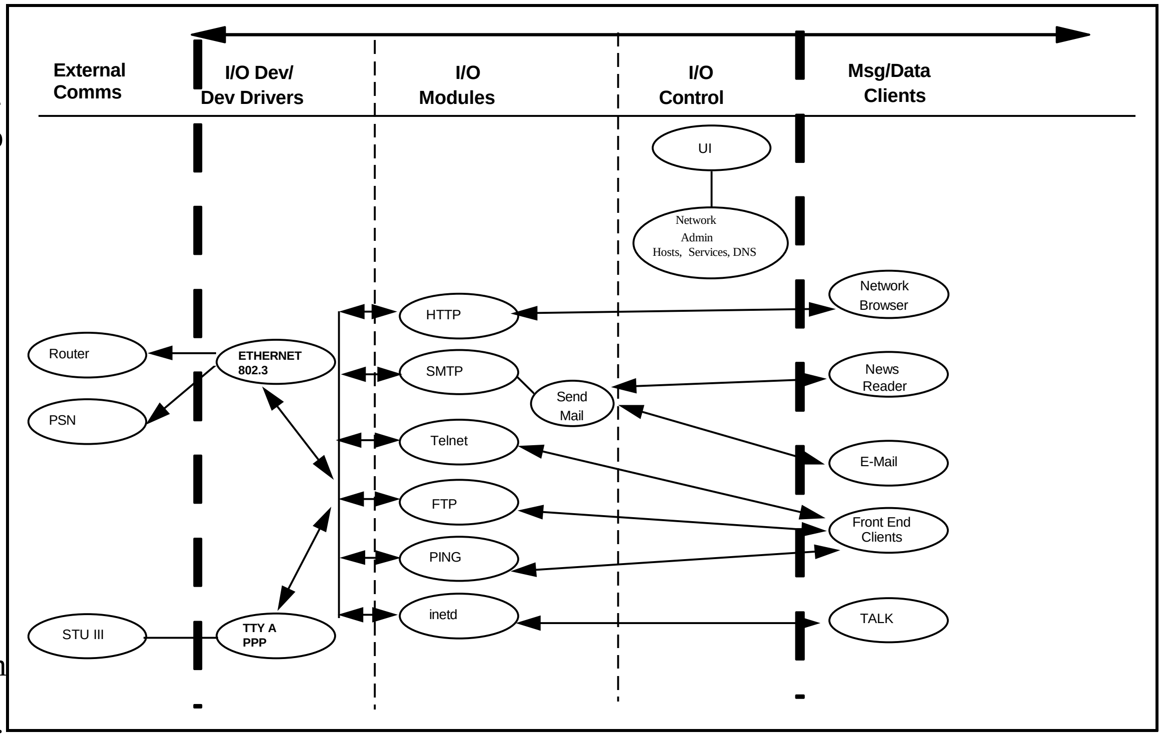


Figure 4-4. COE Notional Server Architecture

Figure 4-4 above provides a view of the notional communications architecture showing the various applications servers. The example provides a view of the communications services available to COE application developers.

4.6.9.1 Message Transfer Services

• Personal E-Mail Messaging	Makes Use of Network Services
• Organizational Messaging	Makes Use of Both Network and Channel Specific Services (e.g., DMS, AUTODIN)
• Tactical Messaging	Makes Use of Channel Specific Services (e.g., TADIL, TACINTEL, OTCIXS, MTS)

4.6.9.2 Examples of Protocols Supported:

- Ethernet (802.3)
- FDDI (Fiber Optics)
- TCP/IP (Transport)
- SMTP (Mail with MIME)
- FTP (File Transfer)
- TELNET (Remote Terminal)
- HTTP (Web Browsers)
- X.400 (DMS)
- X.500 (Directory Services)

4.6.10 Relationships to Other COE Areas

The Communications Services Provides Services to:

- Message Processing (Transfer of Messages, API)
- Multimedia (Transfer of Multimedia Files, IT*)
- Office Automation (Transfer of Electronic Mail, IT*).

Communications COE Utilizes Services Provided by:

- Distributed Computing Services (Directory Services, API)
- Alerts (Presentation of Alerts, API)
- Security (Implementation Guidelines IAW Security SRS)
- Message Queuing (Distributed Applications, OLTP, CPI-C)

* "IT" implies "Information Transfer" using network services and may utilize published communication service API's depending on the level of data being used in applications.

4.6.11 COE Communications - Summary

The DII COE Communications Services provide for 'inter-site' and 'intra-site' communications for use by application developers. Some of the services are:

- Address, Data, Visual, information transfer,
- Voice Communications, remote computing
- Networked applications services
- Network and Channel-Specific servers.

DISA is developing Software Requirements Specifications (SRS) to addresses all COE software requirements for all the DII Communications Services. The communications COE will provide a set of APIs for use by client applications and other COE services. The SRS will emphasize network and tactical communications requirements as well as information transfer.

The infrastructure of coordinated services supporting connectivity and data exchange between one DII or GCCS system/workstation and another will also allow system to be extended and scaled to meet the full spectrum of DoD mission needs.

4.7 COE Auxiliary Services

The DII COE also features a look ahead at the future of the technology and new development tools by providing a set of auxiliary services. These auxiliary services provide fast-paced communication and rapid growth of strategic plans, requirement analysis, and documentation. These auxiliary services also help in timely development, dissemination and enterprise-wide access to information and segments. These auxiliary services offer a wide variety of information representing, viewing and processing across the major corporations, governmental agencies and global enterprises.

These services are available at an on-line software configuration management repository and an on-line information server. These services, with appropriate restrictions, are available to segment developers, program managers, site administrators, services and agencies, and program sponsors.

Several network technologies are used to implement COE on-line services.

World-Wide Web (WWW)	Access to catalogs, segments, plans, documents, etc. is provided via a WWW server. Users will require a Hypertext Markup Language (HTML) browser such as Mosaic or Netscape to access the WWW server.
Internet News	An Internet news server is used to manage news groups for the COE and COE-based systems. Such groups include technical discussions related to COE architecture, available tools, and standards.
List Servers	List servers are used to support users without Internet access, but who have gateways capable of supporting electronic mail. List servers provide news, documents, and software via electronic mail.
Anonymous ftp	Anonymous FTP servers are used to provide rapid dissemination of segments to operational sites. Sites may receive segments in either a “push” or a “pull” mode.
Electronic mail	Automatic notification of key events (segment in test, segment ready for distribution, etc.) trouble reports, and meeting notices is done via electronic mail.

SECTION 5.0 ADMINISTRATION

5.1 Compliance

5.1.1 COE Compliance

The degree to which "plug and play" is possible depends upon the degree to which segments are COE compliant. Following is an overview of the compliance requirements. For more detailed information, see the *Integration and Runtime Specification* (I&RTS, Version 2.0, dated 23 October 1995). Appendix B of the *I&RTS* contains a detailed checklist for areas where compliance is mandatory, and a checklist for areas where compliance is ultimately required but for which there is room for a migration strategy to achieve full compliance. The COE provides a suite of tools, described in Appendix C of the *I&RTS*, which validates COE conformance.

By its very nature, an exhaustive list of "do's and don'ts" is not possible. COE compliance must be guided by overarching principles with checklists and tools to aid in detecting as many problem areas as possible. Full COE compliance embodies the following principles:

2. All segments shall comply with the guidelines, specifications, and standards defined in this document and related documents such as the *DII COE Style Guide*.

2. All segments and data shall be structured in segment format. By definition, COTS components of the bootstrap COE are exempted from this requirement. Segment format is described in *I&RTS*, Chapter 5.

2. All segments shall be registered and submitted to the COE Software Repository System (CSRS). The registration process is described in the *I&RTS*, Appendix E while submission of segments to the CSRS is described in Chapter 7. Also see "*Configuration Management Software and Documentation Requirements*," Version 1.0, June 1996.

2. All segments shall be validated with the *VerifySeg* tool prior to submission, and shall successfully pass the *VerifySeg* with no errors. An annotated listing of the *VerifySeg* output shall be submitted with each application that uses it.

2. All segments shall be loaded and tested in the COE prior to submission. Segment developers are responsible for testing their segment within the full COE, but there is no requirement to include mission application segments for which there is no dependency.

2. All segments shall fully specify dependencies and required resources through the appropriate segment descriptors defined in the *I&RTS*, Chapter 5.

2. All segments shall be designed to be removable, and tested to confirm that they can be successfully removed from the system. Some segments, especially COE

components, are designed to be “permanent” but even those must be removable when a later segment release supersedes the current one.

2. All segments shall only access COE components through the published APIs, and segments shall not duplicate functionality contained within the COE. There is no requirement to integrate to COE functionality which is not required by the segment, but note that use of some segments may have an implicit dependency on other segments.

2. No segment shall modify the environment or any files it does not own except through environment extension files or through use of the installation tools provided by the COE.

COE compliance can also be addressed by the degree to which integration with COE component segments and the runtime environment has been achieved, from “peaceful coexistence” to “fully integrated.” However, the degree of software integration achieved is only one important way of measuring COE compliance. Equally important are measures of GUI consistency, architectural compatibility, and software quality. As shown in Figure 5-1, the DII COE defines four areas of compliance, shown below, called “compliance categories.” Within a specific category, a segment is assigned a numerical value, called the “compliance level,” which is a measure of the degree to which a segment is compliant within the category. The DII COE takes this approach because it is especially useful to have compliance categories and levels when defining a migration strategy for legacy systems.

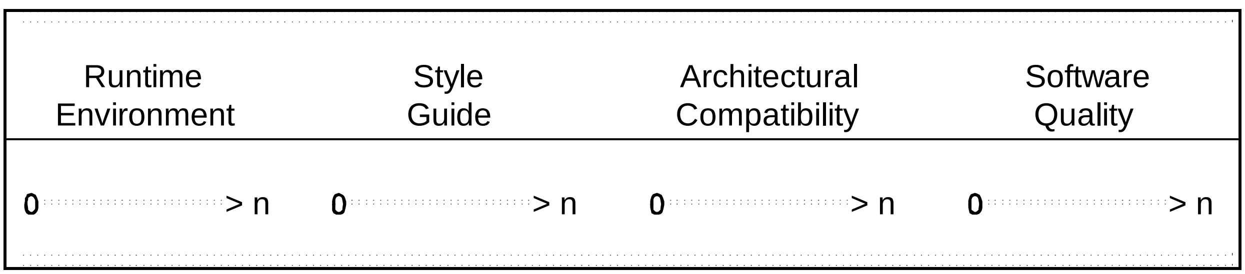


Figure 5-1. COE Compliance Categories and Levels

The four DII COE compliance categories are:

Category 1: Runtime Environment. This category measures how well the proposed software fits within the COE executing environment, and the degree to which the software reuses COE components. It is an assessment of whether or not the software will “run” when loaded on a COE platform, and whether or not it will interfere with other segments.

Category 2: Style Guide. This category measures how well the proposed software operates from a “look and feel” perspective. It is an assessment of how consistent the overall system will appear to the end user. It is important that the resulting COE-based system appear seamless and consistent to minimize training and maintenance costs.

Category 3: Architectural Compatibility. This category measures how well the proposed software fits within the COE architecture (client/server architecture, DCE infrastructure, CDE desktop, etc.). It is an assessment of the software's potential longevity as the COE evolves. It does *not* imply that all software must be client/server and RPC (Remote Procedure Call) based. It simply means that a reasonable design choice has been made given that the COE is client/server based and is built on top of a DCE (Distributed Computing Environment) infrastructure.

Category 4: Software Quality. This category measures traditional software metrics (lines of code, McCabe complexity metric, etc.). It is an assessment of program risk and software maturity.

The COE defines eight progressively deeper levels of integration for the Runtime Environment Category. Note that levels1-3 are "interfacing" with the COE, not true integration. Integration begins at level 4.

The DII COE compliance levels are:

Level 1: Standards Compliance Level. A superficial level in which the proposed capabilities share only a common set of COTS standards. Sharing of data is undisciplined and minimal software reuse exists beyond the COTS. Level 1 may allow simultaneous execution of the two systems.

Level 2: Network Compliance Level. Two capabilities coexist on the same LAN but on different CPUs. Limited data sharing is possible. If common user interface standards are used, applications on the LAN may have a common appearance to the user.

Level 3: Workstation Compliance Level. Environmental conflicts have been resolved so that two applications may reside on the same LAN, share data, and coexist on the same workstation as COE-based software. The kernel COE, or its equivalent, must reside on the workstation. Segmenting may not have been performed, but some COE components may be reused. Applications do not use the COE services and are not necessarily interoperable.

Level 4: Bootstrap Compliance Level. All applications are in segment format and share the bootstrap COE. Segment formatting allows automatic checking for certain types of application conflicts. Use of COE services is not achieved and users may require separate login accounts to switch between applications.

Level 5: Minimal COE Compliance Level. All segments share the same kernel COE, and functionality is available via the Executive Manager. Boot, background, and local processes are specified through the appropriate segment descriptor files. Segments are registered and available through the on-line library. Applications appear integrated to the user, but there may be duplication of functionality and interoperability is not guaranteed. Segments may be successfully installed and removed through the COE installation tools.

Level 6: Intermediate COE Compliance Level. Segments utilize existing account groups, and reuse one or more COE component segments. Minor documented differences may exist between the *DII COE Style Guide* and the segment's GUI implementation.

Level 7: Interoperable Compliance Level. Segments reuse COE component segments to ensure interoperability. These include COE provided communications interfaces, message parsers, database tables, track data elements, and logistics services. All access is through published APIs, with documented use of few, if any, private APIs. Segments do not duplicate any functionality contained in COE component segments.

Level 8: Full COE Compliance Level. Proposed new functionality is completely integrated into the system (e.g., makes maximum possible use of COE services) and is available via the Executive Manager. The segment is fully compliant with the *DII COE Style Guide*, and uses only published public APIs. The segment does not duplicate any functionality contained elsewhere in the system whether as part of the COE or as part of another mission application segment.

5.1.2 Bootstrap Compliance

A developer must achieve Bootstrap Compliance (Level 4) for a segment before it is submitted to DISA for evaluation as a prototype. Such segments will not be fielded nor accepted into the on-line library. At DISA's discretion, segments which meet the criteria for Minimal COE Compliance (Level 5) may be accepted into the on-line library, and installed at selected sites as prototypes for user evaluation and feedback. Such segments are not accepted as fieldable products. Acceptance as an official DISA fieldable product requires demonstration of Interoperable Compliance (Level 7) and a migration strategy to Full COE Compliance (Level 8), unless the proposed segment is an interim product that is targeted to be phased out in the near term.

The compliance categories and levels defined here are a natural outcome of developing a reasonable approach to migrating legacy systems into the COE. The first step of Runtime Environment (Category 1), covered by Levels 1-4, is to ensure that systems do not destructively interfere with each other when located at the same operational site. Level 5 is sometimes called a "federation of systems" in that systems are still maintained as "stovepipes," but they can safely share common hardware platform resources. Levels 6-8 complete the approach by reducing functional duplication, promoting true data sharing, and making the system appear to the user as if it were developed as a single system. The last three levels represent varying degrees of integration from marginally acceptable (Level 6) to a truly integrated system (Level 8).

5.2 Segment Registration

Segment Registration is the entry point into the development process. Its purpose is to collect information about the segment for publication in a *segment catalog*. Perhaps the most difficult part of maintaining a software repository is simply knowing what capabilities exist. This is the purpose of maintaining a DII segment catalog. The segment catalog is available on-line (via SIPRNet) through an HTML browser and contains information provided by developers in a segment registration form Keyword searches can be performed on the catalog by developers to identify reusable segments, or by operational sites to find new mission applications. For information on segment submittal, reference "*Configuration Management Software and Documentation Requirements*," Version 1.0, June 1996.

5.2.1 Segment Descriptor

Segment registration requires the following information about each segment:

Segment Information

- Segment Name (full descriptive title)
- Target System (GCCS, GCSS, COE component, etc.)
- Segment Type (data, software, account group, etc.)
- Segment Prefix
- Segment Home Directory
- Estimated memory required by the segment
- Estimated disk storage requirements
- Platform availability (PC only, Solaris only, etc.)
- Programming language(s)
- List of Related Segments (if applicable)
    - Aggregate components
    - Required segments including version and patch requirements
- Operating System(s) supported
- Windowing Software Required, including versions
- Other Required COTS Products, including versions

Management Information

- Name, Organization, Address, Phone, Fax, and E-mail address for primary and optional alternate Program Management Point of Contact
- Name, Organization, Address, Phone, Fax, and E-mail address for primary and optional alternate Technical Point of Contact
- Name, Organization, Address, Phone, Fax, and E-mail address for primary and optional alternate Process Point of Contact
- Special Restrictions (e.g., not releasable)

Miscellaneous Catalog Information

- Short paragraph describing segment features and purpose
- List of keywords for use in catalog searches
- Unclassified picture of the segment user interface (GIF, JPEG, or X11 Bitmap format) (optional).

Not all information provided at segment registration time is made available to the community at large. The 'technical point of contact' is available only to the DISA Engineering Office in the event that technical questions or issues arise during segment integration. The 'process point of contact' is the individual authorized by the segment program manager to actually submit the segment, or to receive status information and notifications. The 'program management point of contact' is the only individual authorized to commit schedule or resources, and is the only individual authorized to release information about the segment to the community at large. The three points of contact are selected by the service/agency responsible for the segment. Services may elect to designate a single individual for all three points of contact, and may include an alternate point of contact for each category.

Each segment is assigned an identifier called a *segment prefix*. The segment prefix is a 1-6 alphanumeric character string which is used to prevent naming conflicts between segments. Use of the segment prefix is required in any situation where there is the possibility that two different segment developers might choose the same name for a public symbol such as an environment variable, executable, API, or library. Two segments may in fact have the same segment prefix as long as there is no possibility that public symbols will conflict.

Segment directory names are often the same as the segment prefix, but they do not have to be. Directory names can be any directory name that conforms to rules imposed by the operating system, provided they consist only of printable characters, begin with an alphanumeric character, and are not already in use by another segment. It is recommended that directory names be limited to 14 characters to avoid porting problems.

5.2.2 Registration Process

Two steps constitute the Segment Registration phase:

1. Register the segment. The segment registration form can be submitted in written form, through email, or in HTML format. Once the developer submits the registration form, the information is entered into the CSRS and confirmation is sent to the process point of contact. Segment information is entered into the segment catalog with a tentative release date for the segment. The segment prefix and directory requested will be granted unless they have already been assigned to other developers' segment.

2. Download segments required for development. When notification is received that segment registration was successful, developers may download COE component segments, developer's toolkit, object code libraries, and other segments required for software development. Appendix D of the *I&RTS* document provides more information on how to download segments, tools, libraries, etc. It also provides information on how to access and search the on-line segment catalog.

5.3 Configuration Management

Configuration Management comes into play when the developer submits a segment to DISA. Submittal requirements are outlined in "*Configuration Management Software and Documentation Requirements*," Version 1.0, June 1996.

Prior to submitting a component to DISA, a developer must

- package the component as a segment,
- demonstrate COE compliance through tools and checklists,
- test the segment in isolation with the COE,
- provide required segment documentation, and
- demonstrate the segment operating within the COE.

The Software Support Activity (SSA) then enters the segment into the on-line library (the CSRS) for configuration management purposes and confirms COE compliance by running the same suite of tools as the developer. The SSA then tests interaction between segments and the impact on performance, memory utilization, etc. Since segments typically can only interact through the COE, the task is greatly simplified and the need for human intervention in the process is minimized.

5.3.1 COE On-line Software Repository (CSRS)

The COE software repository is used to store and disseminate COE products. The software repository is built on top of the Defense Software Repository System (DSRS) and is accessible only from SIPRNet. Segments, technical documentation, APIs, the COE Developers' Toolkit, and segment abstracts are stored in the repository.

Segments are sent electronically to the DISA Operational Support Facility (OSF) through the submit program. Segments may also be sent to the OSF via tape. Segments are compressed to reduce transmission time, and encrypted to provide security. A daemon running on a system at the OSF receives the segment and places it into a protected directory until it is tested for conformance and to ensure that it is an authorized segment. Only then is the segment actually checked into CSRS. This process is described in more detail in Chapter 3 of the DII COE *I&RTS* document.

Segments are retrieved from CSRS in a similar way. As segments are approved for release, they are placed in a protected directory that is accessible via an 'anonymous ftp', or through a network browser.

Developers who desire CSRS access must request access from DISA through their appropriate government program sponsor. Those without SIPRNet access may request COE products, such as the Developers' Toolkit, on tape media.

Distribution of COE-based systems to operational sites also uses the CSRS. Site administrators must request access from DISA through their appropriate government channels.

5.3.2 COE Information Server

The COE information server (CINFO) is used to disseminate information to the at-large COE community. The information server provides the following types of information:

- general product information
- meeting minutes

- briefings
- segment descriptions
- user documentation
- programmatic documentation
- problem reports.

An unclassified WWW home page available via the Internet provides access only to non-sensitive general information from these categories. The classified WWW home page is available only on SIPRNet and includes a list of all available segments, segment version and patch information, information on upcoming system changes, and special installation instructions.

All information posted on the information server requires prior approval by the DISA Engineering Office. Information to be posted must be submitted to the engineering office by the appropriate service/agency representative.

5.4 Problem Reporting

DISA has established a procedure to report any problems detected in the DII COE and established an Incident Control Center to process incoming problems. Problems are either recorded on a Global System Problem Report (GSPR, DISA form 291) or transmitted via electronic mail. A GSPR is included as Appendix F to this document.

Once received, a configuration manager assigns a control number to the problem and provides the number to the reporter prior to document review. The problem is then sent to a system engineer for evaluation. If the problem has been fixed, the configuration manager is notified and the update is distributed. Otherwise the problem is assigned for action. When completed, the configuration manager is notified and the updates propagated as before.

APPENDIX A:

ACRONYM LIST

Appendix A: Acronym List

ABCS	Army Battle Command System
API	Application Program Interface
APP	Application Portability Profile
ATO	Air Tasking Orders
C4I	Command, Control, Communications, Computers, and Intelligence
CALS	Continuous Acquisition and Life Cycle Support
CDE	Common Desktop Environment
CFCSE	Center for Computer Systems Engineering
CMM	Capability Maturity Model
COE	Common Operating Environment
COTS	Commercial Off-the-Shelf
CSCI	Computer Software Configuration Items
CSRS	COE Software Repository System
DBA	Database Administrator
DBMS	Database Management System
DCE	Distributed Computing Environment
DMS	Defense Message System
DoD	Department of Defense
DII	Defense Information Infrastructure
DISA	Defense Information Systems Agency
ECP	Engineering Change Proposal
FSP	Functional Standards Profile
GCCS	Global Command and Control System
GCSS	Global Combat Support System
GOTS	Government Off-the-Shelf
I&RTS	Integration and Runtime Specification
IDB	Integrated Database
IEEE	Institute for Electrical & Electronic Engineers
IRM	Information Resource Managers
ISP	International Standard Profile
JIEO	Joint Interoperability and Engineering Organization
JMCIS	Navy Joint Maritime Command Information System
JOPES	Joint Operations Planning and Execution System
LAN	Local Area Network

MCG&I	Mapping, Charting, Geodesy, and Imagery
NIST	National Institute of Standards and Technology
NDI	Non-Development Items
NFS	Network File Server
OSE	Open System Environment
OSF	Open Software Foundation
POC	Point of Contact
POSIX	Portable Operating System Interface for UNIX
RDBMS	Relational Database Management System
RM	Reference Model
RPC	Remote Procedure Calls
SEI	Software Engineering Institute
SIPRNET	Secure Internet Protocol Router Network
TAFIM	Department of Defense Technical Architecture Framework for Information Management
TCP/IP	Transmission Control Protocol/Internet Protocol
TPFDD	Time Phased Force and Deployment Data
UDP	User Datagram Protocol
WAN	Wide Area Network

This page intentionally left blank.

APPENDIX B:

GLOSSARY

Appendix B: Glossary

Account Group: A template for establishing a runtime environment context for individual operators. Account groups are typically used to do a high-level segregation of operators into system administrators, security administrators, database administrators, or mission-specific operators.

Affected Account Group(s): The account group(s) to which a segment applies. Functionality provided by the installed segment will normally appear to the operator as new menu items or icons in the affected account group(s).

Aggregate Segment: A collection of segments grouped together, installed, deleted, and managed as a single unit.

Application Program Interface (API): The API is the interface, or set of functions between the application software and the application platform. An API is categorized according to the types of service accessible via that API. There are four types of API services: 1) User Interface Services, 2) Information Interchange Services, 3) Communication Services, and 4) Internal System.

Approved Software: Commercial software products that have been tested as compatible with the COE. In this context, approved software implies only that the software has been tested and confirmed to work within the environment. It does not imply that the software has been approved or authorized by any government agency for any specific system.

Bootstrap COE: That subset of the COE that is loaded in order to have enough of an operational environment that segments can be loaded. The bootstrap COE is typically loaded along with the operating system through vendor-supplied instructions or UNIX commands such as *tar* and *cpio*.

Client: A computer program, such as a mission application, that requires a service. Clients are consumers of data while servers are producers of data.

Commercial Off-The-Shelf Software (COTS): Software that is available commercially. Examples include versions of UNIX, X Windows, or Motif, as well as approved software such as Oracle, Sybase, and Informix.

Common Operating Environment (COE): The architecture, software infrastructure, core software, APIs, runtime environment definition, standards and guidelines, and methodology required to build a mission application. The COE allows segments created by separate developers to function together as an integrated system.

Community Files: Files that reside outside a segment's assigned directory. To prevent conflict among segments, community files may be modified only through the COE installation tools. Examples of community files include: */etc/passwd, /etc/hosts, /etc/services*.

Compliance: A numeric value, called the compliance level, which measures the degree to which a segment conforms to the principles and requirements defined by COE standards, and the degree to which the segment makes use of COE services. Compliance is measured in four areas, called compliance categories. The four categories are Runtime Environment, Architectural Comparability, Style Guide, and Software Quality,

Component Database: Individual databases within a multi-database design.

Configuration Control Board (CCB): The organization responsible for authorizing enhancements, corrections, and revisions to the COE, or to a COE based system.

Database: A structured set of data, managed by a DBMS, together with the rules and constraints for accessing the data.

Database Management System (DBMS): Software to manage concurrent access to shared databases.

Database Schema: The design of a particular database.

Descriptor Directory: The subdirectory *SegDescrip* associated with each segment. This subdirectory contains descriptors that provide information required to install the segment.

Descriptors: Data files (contained in the segment's descriptor directory) that are used to describe a segment to the COE. The software installation and integration process uses descriptor directories and their descriptor files to ensure COE compliance. Descriptor files permit automated integration and installation.

Development Environment: The software environment required to create, compile, and test software. This includes compilers, editors, linkers, debug software, and developer configuration preferences such as command aliases. The development environment is distinct from the runtime environment, and must be separated from the runtime environment, but is usually an extension of the runtime environment.

Distributed Database: A database whose data objects exist across multiple computer systems or sites.

Distributed Processing: The ability to perform collaborative processing across multiple computers. This capability allows processing load to be distributed.

Environment: In the context of the COE, all software that is running from the time the computer is rebooted to the time the system is ready to respond to operator queries after operator login. This software includes the operating system, security software, installation software, windowing environment, COE services etc. The environment is subdivided into a runtime environment and a software development environment.

Environment Extension File: A file that contains environmental extensions for the COE. Segments use extension files to add their own environment variables and other items to the COE.

Fragmentation Schema: The distribution design for a distributed database.

Government Off-The-Shelf (GOTS) Software: Software developed through funding by the U.S. Government.

Kernel COE: That subset of the COE component segments which is required on all workstations. As a minimum, this consists of the operating system, windowing software, security, segment installation software, and Executive Manager.

Multi-Database: A collection of autonomous databases.

Profile: The subset of the total COE-based functionality that is to be made available to a group of individual operators.

Runtime Environment: The runtime context determined by the applicable account group, the COE, and the executing segments.

Segment : A collection of one of more CSCIs (Computer Software Configuration Items) most conveniently managed as a unit. Segments are generally defined to keep related CSCIs together so that functionality may be easily included or excluded in a variant.

Segment Prefix: A 1-6 alphanumeric character string assigned to each segment for use in naming public segments.

Server: A computer program that provides some service. Servers are producers of data while clients are consumers of data.

Service: A function that is common to a number of programs, such as performing some extensive calculation or retrieving a category of data.

Session: An individual connection between an application program and a database management system.

Superset: The sum total collection of all COE based segments available to the development community. The superset includes the COE as well as all mission-application segments.

Variant: A subset of the superset of all software. This subset includes the COE, and is fielded to service an operational mission area. A variant represents that collection of segments, including COE component segments, that are suitable for a particular site, mission area, or workstation. See also the definition of mission area, site, system, and workstation variants.

Workstation Variant: A collection of segments as installed and configured on a particular workstation.

APPENDIX C:

REFERENCE DOCUMENTS

Appendix C: Reference Documents

Reference Document Title	Version
Design and Concept Documents	
Technical Architecture Framework for Information Management (TAFIM)	Version 4.0
Architectural Design Document for the Defense Information infrastructure (DII) Common Operating Environment (COE)	Draft January 1996
JLSC Integrated Technical Architecture and Common Operating Environment	Version 1.0 October 30, 1995
GCCS Baseline Common Operating Environment	Version 2.1 July 31, 1995
Transition Approach for the Defense Information Infrastructure (DII) Common Operating Environment (COE)	10/9/95 Draft
Global Command and Control System (GCCS) System and Network Management Concept of Operations (CONOPS)	Version1.6 14 September 1995
Requirements and Specifications	
User Interface Specifications for the Defense Information Infrastructure (DII)	Version 2.0 Preliminary Draft December 31, 1995
GCCS Common Operating Environment Requirements	August 15, 1994
Global Command and Control System (GCCS) Common Desktop Environment (CDE) Integration specification	Version 1.0 December 8,1995
Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specifications (I&RTS)	Preliminary Version 2.0 10/23/95
Specification for DII Version 1.0 COE Kernel	
SRS Documents	
Command and Control System (GCCS) Common Operating Environment (COE) System Administration Software Requirements Specification	August 21, 1995 v1.0
Software Requirements Specification (SRS) for the Global Command and Control System (GCCS) Communication Services	November 1995
Security Software Requirements Specification (SRS)	Version 1.2 1 December 1995
Software Requirements Specification for the Network Administration Functional Area of the Global Command and Control System (GCCS)	Draft Rev 1.0 29 September 1995
Programmer Guides and Tools	

Reference Document Title	Version
Distributed Computing Environment (DCE) Applications Programming Guide for GCCS	Version 3.0 Draft December 1, 1995
Initial COE Tools List	
JCALS Infrastructure Product Description	1995 Draft
Application Integration Reference Manual JCALS Developer's Tool Kit	1995 Draft
DII COE Documents	
JMTK 1.0.0.3 Developers Manual - Part 2. (Man Pages)	TL-161-06-02 4/8/1996 Version 2.0.0.3
Defense Information Infrastructure (DII) Common Operating Environment (COE) Programming Guide	DII.2.0.0.1DRAFT.UNIX.PG-1 Version 2.0.0.1 Draft (HP and Solaris) June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) Programming Guide	DII.2.0.0.1DRAFT.NT.PG-1 Version 2.0.0.1 Draft (Windows NT) June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) Kernel Installation Guide	DII.2.0.0.1DRAFT.Sol24.IG-1 Version 2.0.0.1 Draft (Solaris 2.4) June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) Kernel Installation Guide	DII.2.0.0.1DRAFT.TAC3.IG-1 Version 2.0.0.1 Draft (HP TAC-3 Workstations) June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) Kernel Installation Guide	DII.2.0.0.1DRAFT.TAC4.IG-1 Version 2.0.0.1 Draft (HP TAC-4 Workstations) June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) Kernel Installation Guide	DII.2.0.0.1DRAFT.NT.IG-1 Version 2.0.0.1 Draft (Windows NT) June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) System Administrator's Guide	DII.2.0.0.1DRAFT.UNIX.SAG-1 Version 2.0.0.1 Draft (HP and Solaris) June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) System Administrator's Guide	DII.2.0.0.1DRAFT.NT.SAG-1 Version 2.0.0.1 Draft (Windows NT) June 14, 1996

Reference Document Title	Version
Defense Information Infrastructure (DII) Common Operating Environment (COE) Application Programmer Interface (API) Reference Guide	DII.2.0.0.1DRAFT.RG-1 Version 2.0.0.1 Draft (HP and Solaris) June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) DII COE User Profiles Application Programming Interface v2.0.0.1	LL-400-27-01 Version 2.0.0.1 Document 4/4/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) SPARCstorage Volume Manager 2.1 Installation Guide	DII.2.0.0.1DRAFT.SAA.IG-1 Version 2.0.0.1 Draft June 14, 1996
Defense Information Infrastructure (DII) Common Operating Environment (COE) TIVOLI Management Environment Segment Version Description Document	LL-216-05-01 Version 2.0.0.1 4/29/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) DII 2.0 Internet Relay Chat Server and Client Installation Procedures-Quick Guide	LL-400-39-01 Version 2.0 Instructions 4/23/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) EMPIRE UNIX Systems Management Agent VDD,SRS,OM,STP,STD,KPL,II for HP	LL-514-02-06a 4/11/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) EMPIRE UNIX Systems Management Agent VDD,SRS,OM,STP,STD,KPL,II for Solaris	LL-514-02-07b 6/12/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) HP NetMtrix Power Agent VDD,SRS,OM STP,STD,KPL,II	LL-515-02-06a Release Notes 4/11/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) HP NetMtrix Power Agent VDD,SRS,OM STP,STD,KPL,II	LL-515-02-07b Release Notes 6/12/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) DCE DCES (DCE Server) Segment v1.0.0.0 Installation Instructions for Solaris	LL-524-19-01a V1.0.0.0 Instructions 4/17/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) DCE DCES (DCE Server) Segment v1.0.0.0 Installation Instructions for HP	LL-524-19-01b V1.0.0.0 Instructions 4/23/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) CMP Application Programmer Interface (API) v 1.2.1.1/Solaris 2.4	LL-526-12-01 V1.2.1.1 Interface 5/28/96

Reference Document Title	Version
Defense Information Infrastructure (DII) Common Operating Environment (COE) CMP Software Version Description (SVD) v 1.2.1.1/Solaris 2.4	LL-526-12-01 V1.2.1.1 Software Version Description 5/28/96
Joint Interoperability Engineering Organization (JIEO) Global Command and Control System (GCCS) HP STREAMS: Kernel Rebuild, Reboot VDD,SRS,OM STP,STD,FPL,II/Copy	LL-537-02-01 4/11/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) DII COE 2.0 Mail Services Installation Procedures-Quick Guide	LL-400-55-01 Guide 6/14/96
Joint Interoperability Engineering Organization (JIEO) Global Command and Control System (GCCS) Netscape NS-NEWSS Installation Procedures-Quick Guide	LL-511-08-01 Guide 6/17/96
Joint Interoperability Engineering Organization (JIEO) Global Command and Control System (GCCS) Netsite Web Server and Netscape Web Browser Client Installation Procedures	LL-519-04-01 Guide 6/14/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) DII COE Bootstrap Kernel v2.0.0.1/HP and Solaris Version Description Document	LL-400-06-04 VDD 6/17/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) DII COE Version Description Document for Developers Toolkit/HP and Solaris	LL-400-03-06 VDD 6/17/96
Defense Information Infrastructure (DII) Common Operating Environment (COE) Newsprint Segments NEWSP 2.5/2.0.0.0 and NEWSPR 1.0.0.0 Installation Instructions	LL-405-02-01 Instructions
Defense Information Infrastructure (DII) Common Operating Environment (COE) DII COE 2.0 Kernel Version Description Document for windows NT	LL-400-49-01 VDD 4/24/96
PERL Installation Procedures - Quick Guide	LL-531-03-01 June 14, 1996

This page intentionally left blank.

APPENDIX D:

ON-LINE INFORMATION

Appendix D: On-Line Information

Document or Area	Location
Auxiliary Services	Netscape homepage (http:/www.netscape.com)
Common Desktop Environment	TriTeal Enterprise Desktop homepage (http://www.triteal.com/TED_htmls) Common Desktop Environment overview(http://www.cfi.org) Style Guide
Common Operating Environment	DISA homepage (http://www.disa.mil)
Communications	
Database Environment	I&RTS. Section 4 - COE Database Concepts
Distributed Computing Environment	Transarc homepage (http://www.transarc.com:80/afs)
Interface and RunTime Specification (I&RTS)	DISA homepage (http://www.disa.mil)
Security	Security Software Requirements Specification I&RTS Section 5.8
The Defense Information Infrastructure	DISA homepage (http://www.disa.mil)
User Interface Specifications for the Defense Information Infrastructure (Style Guide)	DISA homepage (http://www.disa.mil)